Accessing Spatial Data using the OGC API Features Interface

(working draft, December 02, 2023)

1. Overview

In this tutorial you will learn how to access geospatial data using web services that implement the OGC API Features interface.

- We will use docker images to install and run an OGC API Features server on our local computer providing data on nature reserves
- We will interact with this service instance by using a browser
- We will use QGIS to demonstrate that off-the-shelf software is capable of interacting with this type of web service in plug-and-play mode.
- We will use a Python Jupyter Notebook to show how you can write programming code that interacts with OGC API Features

The tutorial is structured as follows:

1. Overview
2. Background
   - Feature Data and Feature Services
   - Advantages of accessing data with OGC Web Services
   - OGC-API series of interface specifications
   - OGC-API Features interface
3. Practical exercises
   - Use cases and technical setting
   - Installing software and data
   - Using the web browser to interact with the OGC-API Features service
   - Using QGIS to access feature data via the OGC API Features interface
   - Using Python to interact with the OGC-API Features service
4. Summary and Discussion

This tutorial is designed for students and professionals who want to improve their understanding of Spatial Information Infrastructures. We assume that you have some basic knowledge about geospatial data, QGIS and spatial data analysis. In this case you will need about 30 minutes to use this tutorial.

This Tutorial has been developed in the context of the OER4SDi project at the Institute for Geoinformatics, University of Münster. Authors are Soumya Ganguly and Albert Remke.

tutorial can be used under the terms of the MIT license. Please see the full license terms: https://github.com/oer4sdi/OER-DataAccessVia-OGC-API-Features/blob/main/LICENSE.md) The OER4SDI project has been recommended by the Digital University NRW and is funded by the Ministry of Culture and Science NRW.”.

2. Background

2.1 Feature Data and Feature Services

Real world objects such as roads, buildings, rivers, and land parcels can be represented as Features. Features that share a set of properties can be categorized as Feature Types. One of these properties may be the Geometry that represents the spatial location and extent of the Feature. The Geometry itself may be modeled in various ways, e.g. as points, lines or polygons and is based on a specific Spatial Reference System (SRS). Sets of features of the same feature type can be provided as feature collections.

The ISO 19000 Series of international standards provides a conceptual model of Features and Geometries as well as a set of encoding rules that can be used to create Feature Data for example as XML/GML data.

Since many applications don’t need the rich capabilities of the full fledged ISO data model for geographic information ISO 19125 defines a simplified set of rules for modeling and working with “Simple Features”. As an alternative to XML/GML, many applications use GeoJSON for a lightweight JSON encoding of Simple Feature data.

If the ISO Feature model is used for modeling concrete Feature Types such as roads, rivers, and buildings, the data models are called Application Schemas of the general feature model. They are often defined by textual documents and UML diagrams and come along with XML schema files to support the validation of XML encoded Feature data.

Access to Feature Data can be provided in various ways. One way is to distribute the data in the form of a dataset that is encoded e.g. as XML/GML, GeoJSON, Shape or any other format. Another way is to provide a data access service, i.e., a set of operations, accessible through an Application Programming Interface (API) that supports selective download of the data, data transformations or even manipulation of the data source.

Commonly used APIs are the OGC Web Feature Service interface (OGC WFS) and the OGC API Features interface.

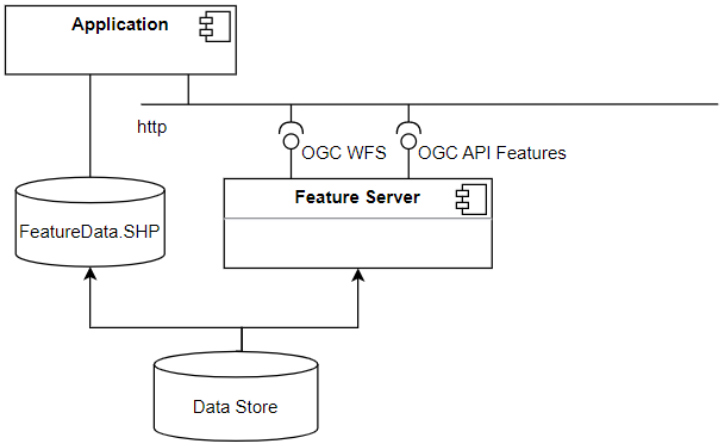

Fig. 1 Access to Feature Data via datasets and services

2.2 Advantages of accessing data with OGC Web Services

Using OGC Web services such as Web Feature Service (WFS) and OGC API features for data exchange offers several advantages over simply transferring prepackaged data files.

- **Higher level of automation**
  OGC Feature Services adhere to established standards that support automated machine-to-machine interaction. This can be used to create software such as QGIS and ArcGIS that largely hide the technical complexity of service interactions and allow users to work with these data sources in a plug an play manner. Furthermore, the data sources can be integrated into fully automated processing chains with complex read and write transactions.

- **Selective data access**
  Each time the FeatureService is accessed, the server delivers the data as requested. Users can retrieve specific subsets of data filtered by spatial, temporal, or attribute properties. In the case of file download, one must first download the file and then filter out the relevant data from it in further processing steps.

- **Efficient delivery of various data products**

Usually, the response to a data request is created on-the-fly according to the request parameters. This allows the data to be provided in different spatial reference systems, encodings, and even variations of the data model. In the case of file download, all variants would have to be offered as separate files.

- **Better support of dynamic data**
  By using Feature Services, you always access the current state of the database as maintained by the provider. Especially for highly dynamic data that needs to be made available in near real-time, using web services is usually more efficient than offering files for download.

- **Data manipulation**
  Feature Services can not only be used to retrieve data but to manipulate the data source as well. Transactional feature services support insert, update and delete transactions on the database so that remote systems can be used to populate and maintain the data source.

2.3 The OGC-API series of interface specifications

So, what is the difference between the older OGC WFS and the newer OGC API Features interface?

The service-oriented architecture of spatial data infrastructures is based on a set of OGC/ISO standards that have been developed since the late 1990s. The OGC WFS belongs to the series of classic OGC Web Services, such as the Web Map Service (WMS), the Web Coverage Service (WCS) or the Catalog Service for the Web (CSW). These services typically use SOAP and http to invoke server side operations such as GetCapabilities, DescribeFeatureType, and GetFeature. These remote procedure calls (RPC) results in response messages with the requested data as payload. This type of OGC interfaces has become widely established over the last 20 years. Today, it is possible in almost every GIS to connect to data directly via WMS and WFS interfaces. However, the interface is complex and not easy to implement.

In contrast to the RPC style of classic OGC web services implementations, the newer OGC API specifications implement the more lightweight RESTful style for web services. One of the key characteristics of RESTful webservices is that http is used as an application protocol, i.e. the semantics of the http methods such as GET, PUT, POST and DELETE are being used to interact with the server side resources rather than just transferring remote procedure calls. Furthermore, the OGC API specifications follow additional recommendations of the World Wide Web consortium (W3C) that have been published as Spatial Data on the Web Best Practices. For example: “Make your spatial

data indexable by search engines" which refers to providing an HTML landing page and crawlable links to metadata and content.

2.4 OGC-API Features interface

OGC API Features is a multi-part interface specification from the Open Geospatial Consortium designed to support the creation, manipulation, and querying of feature data on the web. It specifies requirements and recommendations for RESTful APIs that provide a standardized way to share feature data.

Any service that implements the OGC API Features interface provides you with a standardized set of resources that can typically be accessed in various encodings such as HTML, XML, or JSON:

- **../**
  The **Landing Page** of the service instance is a starting point for exploring the capabilities and data offerings of the service. It contains general information on the service offering, links to the API definition and to the data itself, i.e. the feature collections and feature data.

- **../conformance**
  Starting from the landing page, this path provides access to a declaration of the conformance classes that are supported by the service instance. Each conformance class stands for a set of capabilities described in the OGC -Specs, not all of them are mandatory. This information is important so that systems that want to access the service can easily adjust to its capabilities.

- **../api**
  As said, the landing page must provide a link to the API definition, which provides you with detailed metadata on the capabilities of the service instance such as supported paths and data types. As an option, the service definition may be provided as a sub-resource of the landing page at the path /API. If the service supports the OpenAPI requirements class "oas30" the service definition conforms to the OpenAPI specifications in the version 3.0 (https://www.openapis.org/). But the service definition could be provided in any other format as well, e.g. as an OWS Common capabilities document or as a WSDL document.
  The API definition is needed by developers and development tools to support the implementation of servers and clients. Furthermore it enables standard software such as QGIS and ArcGIS to connect to the server in a plug-and-play manner.

- **../collections**
  The path /collections provides access to the list of feature collections and to the

metadata of each feature collection such as ID, title, description, spatial and temporal extent, schema, and license. Each of the collection provides also links to the items of the feature collection, i.e. to the feature data itself.

- **../collections/{collectionID}**
  Each collection and all of its metadata must be accessible at the path /collections/{collectionID} as well. I.e., this path provides you with a landing page of a certain feature collection.

- **../collections/{collectionID}/items**
  The /items path is the access point to all feature data. The server responds to a GET request at this endpoint by returning a set of feature data. Paging mechanisms are used to limit the amount of data returned in a single request. Query parameters can be used to filter the result set based on spatial and temporal extent or by other feature properties.

- **../collections/{collectionID}/items/{itemID}**
  The ../collections/{collectionID}/items/{itemID} path can be used to access the data of any single feature in the dataset.

The OGC API Features specification does not prescribe any particular encoding for the data provided by a service instance. Currently, the standard specifies four conformance classes: HTML, GeoJSON, GML Simple Features Level 0, and Level 1. I.e., the service instance can implement and declare to conform with some of these conformance classes. In particular, it is recommended that HTML and GeoJSON be supported because they are commonly used to deliver geospatial data on the Web. HTML is important for making geospatial data crawlable and indexable by search engines and for providing easy access to content for human users. GeoJSON is easy to understand and well supported by tools and software libraries.

2.5 Accessing Geospatial Data with OGC API Features

Well, first of all you need to find an instance of such a service to access it. You can use the geoportals of existing SDIs like the Infrastructure for Spatial Information in Europe (INSPIRE), the German national SDI (GDI-DE) or the SDI North Rhine-Westphalia (GDI-NW) to search for datasets that can be accessed via the OGC API Features interface. You will notice that most of the feature services that are offered today still implement the classic OGC WFS interface. However, this will change in the near future and it is expected that OG API features will replace the WFS in the coming years.

Let's assume you have searched for and found a dataset on building footprints and want to access the data. Just use the browser to access the service landing page and browse the service instance paths to view the descriptions and access the collections and the data itself. That is, at this stage, all you need is the browser.

Now, what if you want to integrate the data into your GIS to use it for data analysis and visualization in combination with other resources? Simply use the standard functionality of your GIS software to connect to this type of service by specifying the URL of the landing page. The GIS software will be able to interact with the service based on the interface specification and metadata offered by the service itself (service type, conformance statements, API definition, list of collections, schemas, encodings, etc.).
I.e. you can work directly with the data source, in a similar way as with data sources on your local computer.

In case you are a software developer and want to integrate the data source into your own software solutions, there are many libraries available that support connecting to and interacting with an OGC API feature web service based on the http protocol and the OGC interface specification.

In the next section of our tutorial, we will deepen our freshly acquired knowledge of "OGC API Features" by experiencing the practical use of this type of feature services.

3. Practical exercises

3.1 Use cases and technical setting

In our example, we will demonstrate how to use data on nature reserves via an OGC API Features Service. We will focus on the following use cases:

- Accessing the OGC API Features Web Service from a Browser
- Accessing the OGC API Features Web Service with QGIS
- Accessing the OGC API Features Web Service with Python

We'll use two datasets on protected sites in Germany, which are reusable under the terms of the Data License Germany Attribution 2.0: https://www.govdata.de/dl-de/by-2-0.

a) **NSG - dataset on nature reserves (Naturschutzgebiete),**
provided by the Lower Saxony State Agency for Water Management, Coastal and Nature Conservation (NLWKN, www.nlwkn.niedersachsen.de), Reference: https://registry.gdi-de.org/id/de.ni.mu.nlwkn.csw/EE85FE8F-BD05-4A6D-813B-6ABC4514B18B

b) **Biosphere Reserves (Biosphärenreservate),**
provided by the Bavarian State Office for the Environment (LFU, www.lfu.bayern.de), Reference: https://registry.gdi-de.org/id/de.by/DEBY_4CD25ABA-4ED4-43C0-9A7A-F30D7920846F

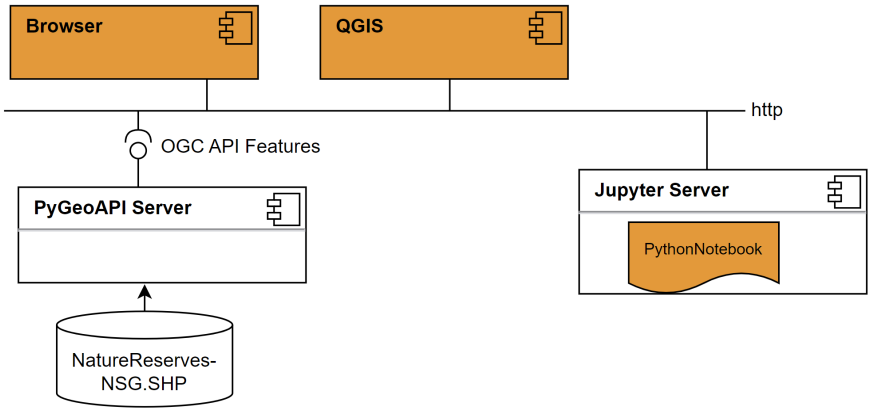

Fig. 2: Technical components and interfaces used in the practical exercise

Figure 2 shows the technical components and interfaces that we will use in our exercise. The protected sites data is being served by a PyGeoAPI server (https://pygeoapi.io/) that provides access to the data via the OGC API Features interface. We use a Browser and QGIS for the first two use cases. For demonstrating how to access the API from a Python script we'll use a Python notebook that runs on a Jupyter server (https://jupyter.org/).

Before we can use these components we have to install the data and the software on our local computer.

3.2 Installing software and data

3.2.1 Check and update QGIS installation

Please make sure that you have a current version of QGIS (version 3.32 or higher) installed on your computer. If not, please visit the QGIS project website https://www.qgis.org and download and install the latest version of QGIS.

3.2.2 Download data and code from the tutorial's GitHub repository

We've prepared the NSG data as well as some Python code and docker configuration files in the GitHub repository of this tutorial:
https://github.com/oer4sdi/OER-DataAccessVia-OGC-API-Features

For downloading the relevant content from the GitHub repository please use one of the following options:

a) If you are experienced in using GIT just open a terminal or command prompt, navigate to the directory you want to use as a WorkingDirectory for this tutorial, and clone the repository with the following command:
git clone https://github.com/oer4sdi/OER-DataAccessVia-OGC-API-Features.git

b) Otherwise go to the GitHub repository, locate the "code" button and select "Download ZIP" to download the zip archive of the repository. Unzip the repository to the location that you want to use as a WorkingDirectory for this tutorial.

3.2.3 Installing the Docker Environment

As to provide a safe runtime environment for our PyGeoAPI and Jupyter servers with all the software and data in place we will use pre-configured docker images that can be installed and executed as docker containers in the docker environment on your local computer. Don't worry if you're not already familiar with Docker, we'll lead you through the installation process step by step.

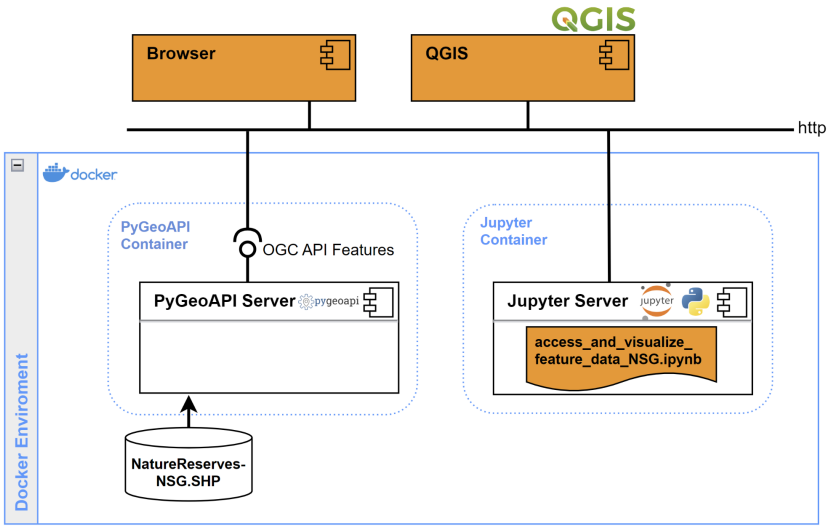

Fig. 3: Docker configuration on your local computer

In our Docker Environment, we will use two docker containers:

a) PyGeoApi container

This container will be used to run the PyGeoAPI Server that provides access to the NSG data using the OGC API Features interface.

The container uses the latest version of the GeoPython PyGeoAPI Docker image. The resource section of the local.config file refers to the data to be published via the OGC API features interface. It overwrites the default configuration file used by the PyGeoAPI Docker image. The data folder containing the NSG data is also loaded into the Docker environment.

b) Python Jupyter Notebook container

This container will be used to run the Jupyter Server that provides the environment for interpreting our Python Notebook. The server and the Python Notebook can then be accessed with a standard browser.

We used a minimal Python docker image that we extended by some libraries for data management and data visualization. These dependencies are defined in the requirements.txt file that is used in the docker file which itself is being used to build the Docker image. A volume reference for an app folder contains the python notebook that you will use to access and visualize the NSG data.

Now let's set up the Docker Environment with the docker images and docker containers step by step:

1) Download, install and start up Docker Desktop

   Visit the Docker website and download the Docker Desktop installer for your operating system. Run the installer which will guide you through the installation process. Once the installation is complete, please start up the Docker Desktop application.

2) Compose and build the docker environment

   Open a command window (terminal) and move to the workingDirectory that contains the unzipped download of the tutorial's GutHub Repository (see section 3.2.2).

   Type **docker-compose up --build** to build and start the docker environment.

After successfully building and launching the docker containers the command window should look like this:

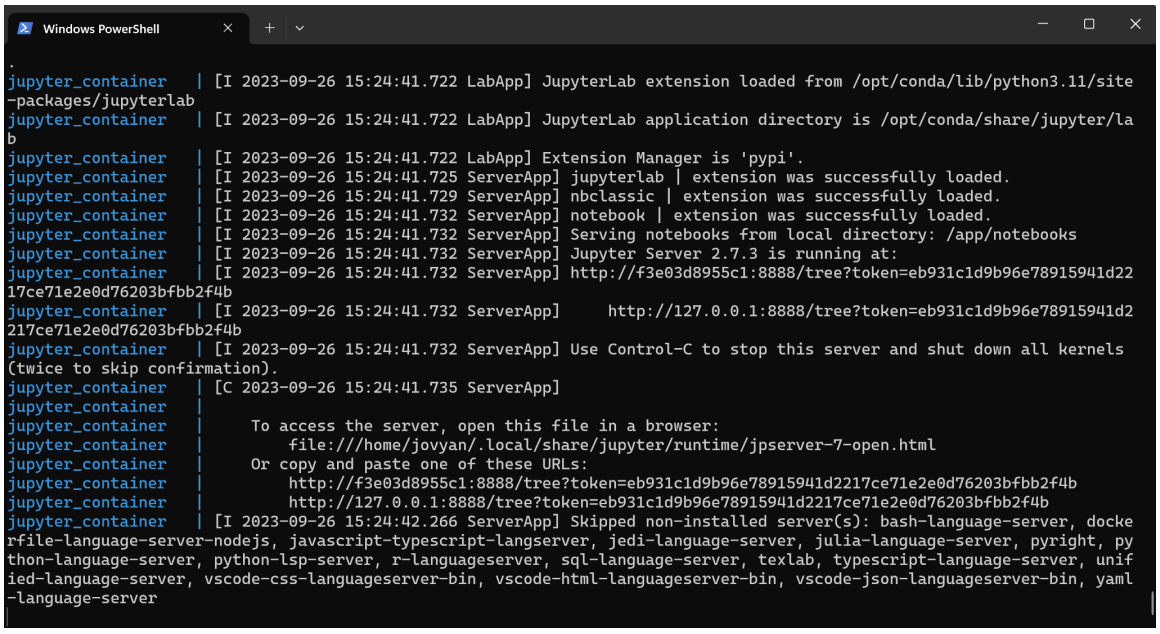

Fig. 04: Docker Compose Logs

Please pay special attention to the logs related to the jupyter_container with information about the URL that can be used to access the Jupyter server:

“http://127.0.0.1:8888/tree?token=.....”

The URL contains a token that the Jupyter server requires as a credential to allow access. To access the Jupyter server, you must copy and paste the URL into the browser window.

The token that is provided with the URL is generated dynamically each time you start the Jupyter server. That is, as soon as you restart the server, you must check the jupyter_container logs to retrieve the URL again with the new token.

The following video briefly demonstrates how to set up the docker environment and how to check if all components are up and running properly.

Video (YouTube): Installing the Docker Environment

3.3 Using the web browser to interact with the OGC-API Features service

In this first exercise, we will explore how to access the OGC API Features Service instance with NSG data using just a standard browser. You will notice that the service provides us with a user-friendly HTML interface, making it simple to navigate through the available feature data.

Let us follow a step by step process to understand how it works.

1. Once the docker containers are up and running, the PyGeoAPI service instance will be available on your local system. Use the URL http://localhost:5000/ in your browser to access the landing page of the NSG service instance.

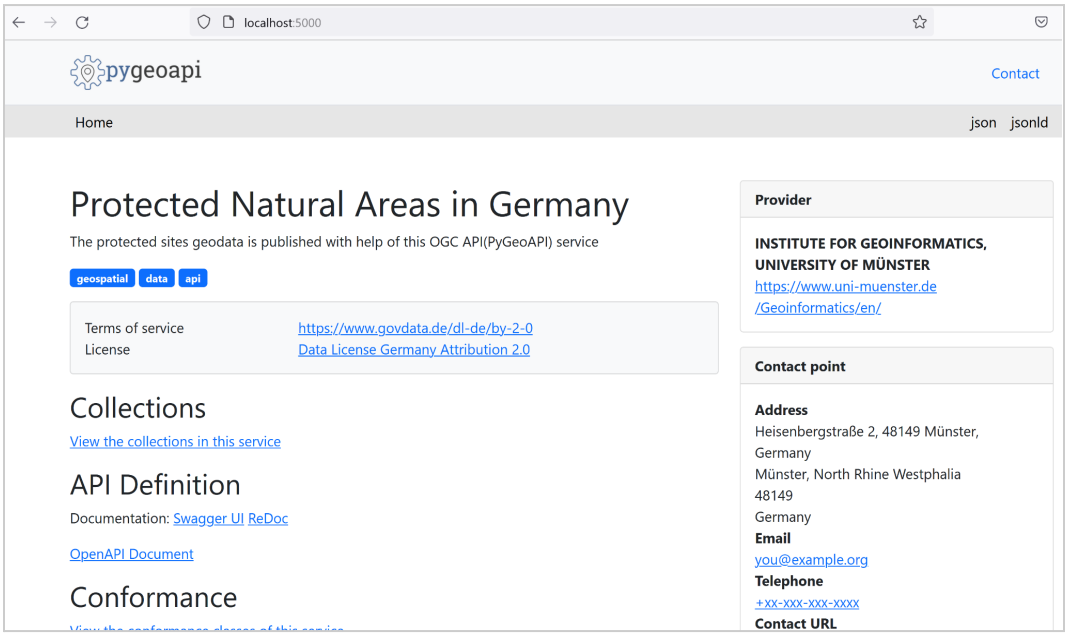

Fig. 05: OGC API Features NSG - Landing Page

This would be the place to provide a brief description of the data so that users know what they can expect from this service. Then you can use the links to look up more details on the services capabilities as well as to dive into the data.

2. Now move to the Collections resource by clicking **“View the collections in this service”.** In the collections page you will see a list of the feature collections that can be accessed via this feature service.

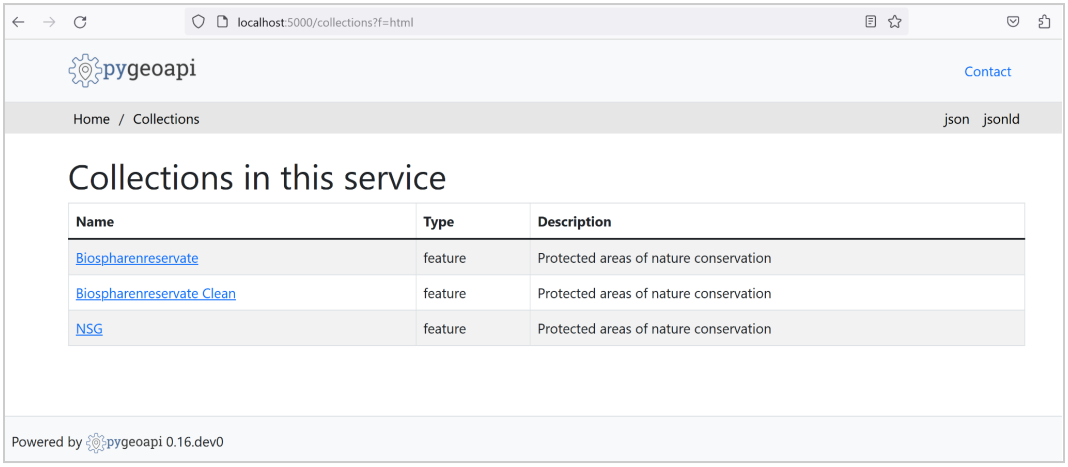

Fig. 06: OGC API Features NSG - Collections

3. Let's take a closer look at the NSG feature collection (Naturschutzgebiete / Nature Reserves). The service offers an HTML representation which comes with an interactive map that visualizes the spatial extent of the dataset.

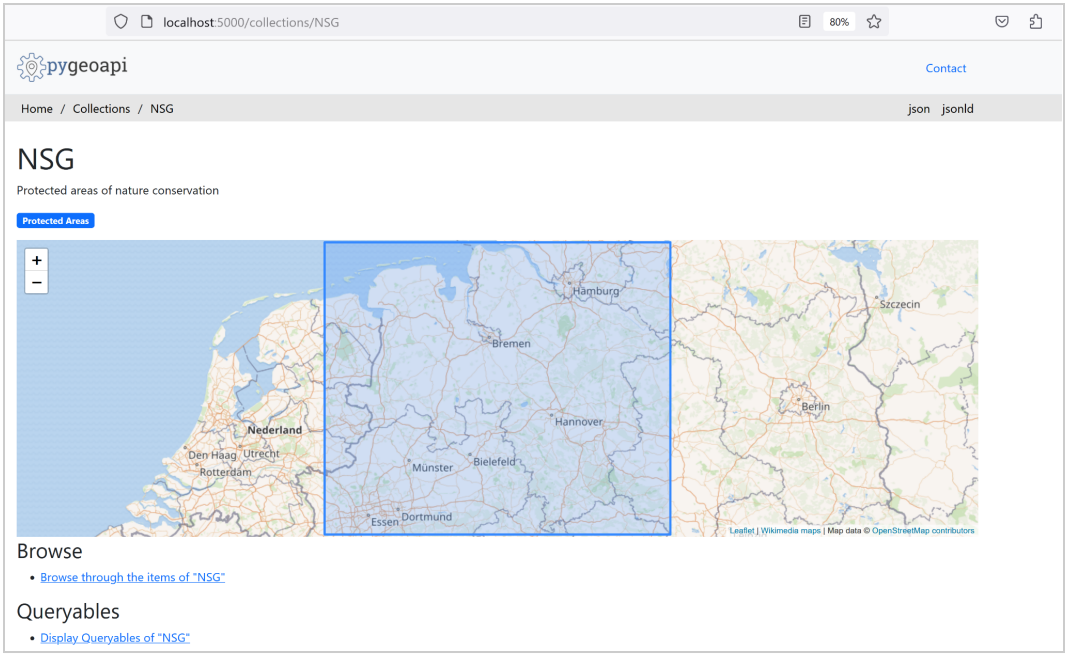

Fig. 07: OGC API Features - Collection NSG

4. Use the “**Browse through the items of NSG” option** to browse through the features in the collection and to visualize their Geometries.

   You can use the interactive map to look up the features in their geographic context. The table to the right lists the properties of all the features. This is already much more informative than just looking at the raw data in a GML or GeoJSON encoding. You can even inspect the location and spatial extent of single nature reserves in detail.

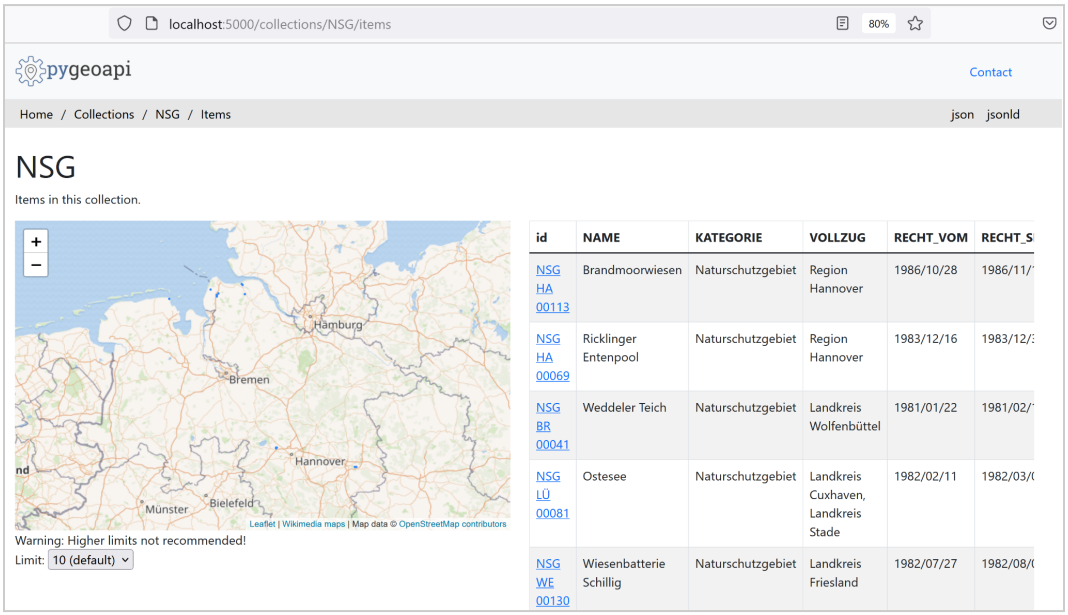

Fig. 08: OGC API Features - Collection NSG, Items

5. Just below the "Browse" option on the NSG feature collection page, you will see the "Queryables" option. Select "Display Queryables of NSG" to view the feature properties and their data types that can be used to filter out features based on their property values.

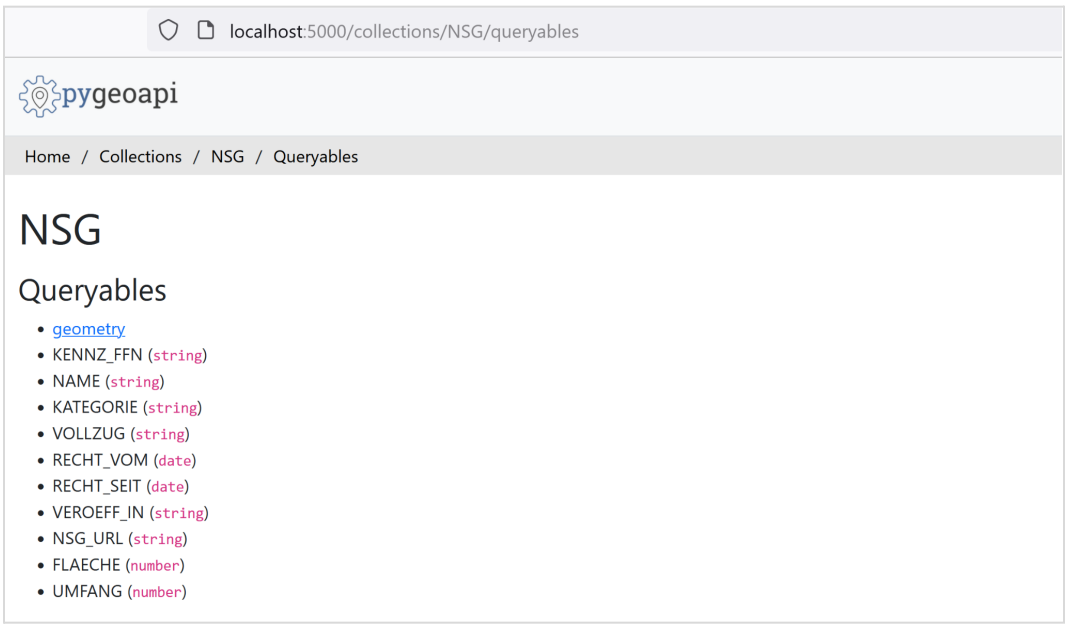

Fig. 09: OGC API Features - Collection NSG, Queryables

6. OK, so “NAME” is a queryable attribute. Let’s try to query all features that have the name “Osteseel”. You can dos so by using the path to the items of the NSG feature collection and add the query “NAME=Ostesee”:

   http://localhost:5000/collections/NSG/items?NAME=Ostesee

   The result should look like this:

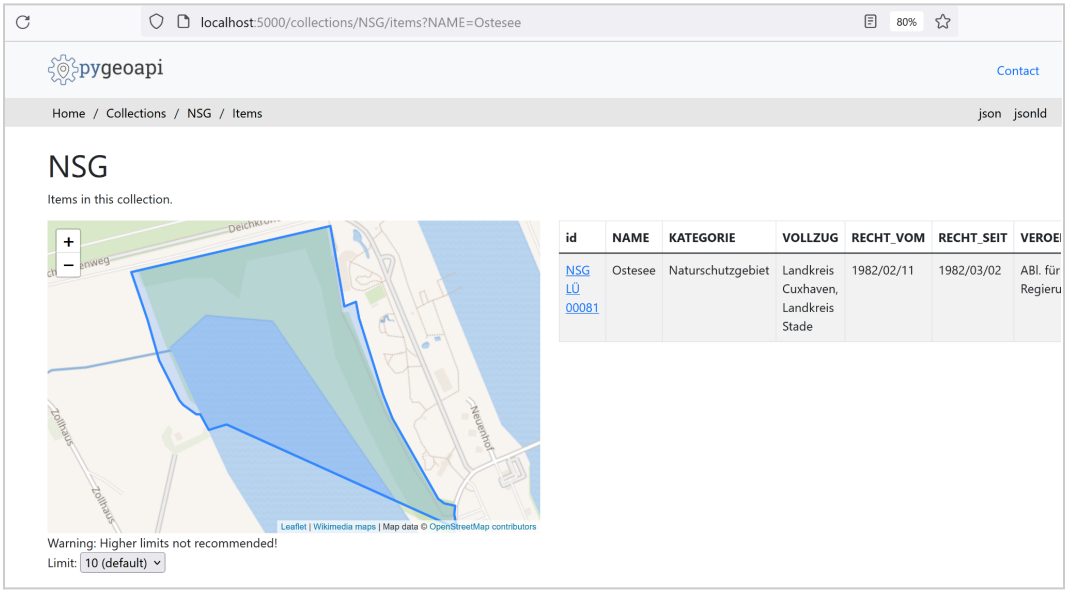

Fig. 10: OGC API Features - Collection NSG, Querying by property values

What we see is the HTML representation of the resource: NSG feature “Osteseel”.

7. Now let's look at the JSON representation of the same feature. Just click on the “json” option in the HTML document or add “f=json” as a second query parameter to the URL. The result should look like this:

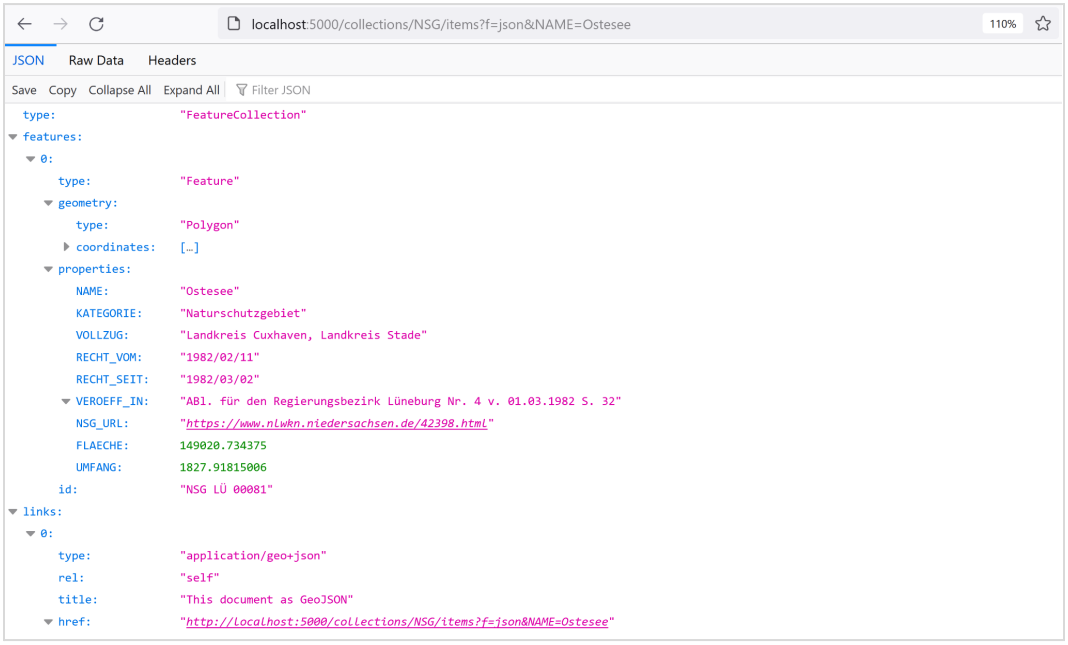

Fig. 11: OGC API Features - Collection NSG, selected Item in JSON encoding

What we can take from this exercise is that the OGC API interfaces support direct interaction with human users, which greatly improves the user experience. Users can effortlessly browse and retrieve geospatial data without complex queries or technical knowledge.

The following video gives a short live demo of the described workflow.
Video (Youtube): Browser Access to Feature Data via the OGC API Features Interface

3.4 Using QGIS to access feature data via the OGC API Features interface

Now we will explore how to access and visualize the NSG data from our OGC-API feature service with QGIS. We will again use the service instance that we set up with PyGeoAPI on our local machine.

1. Firstly, we need to copy the url of the OGC API service which is accessible from the browser. It is `http://localhost:5000/` .

2. Next, we Launch QGIS and select the "Add Layer" button in the toolbar, or go to "Layer" > "Add Layer" > "Add WFS / OGC API Features Layer".

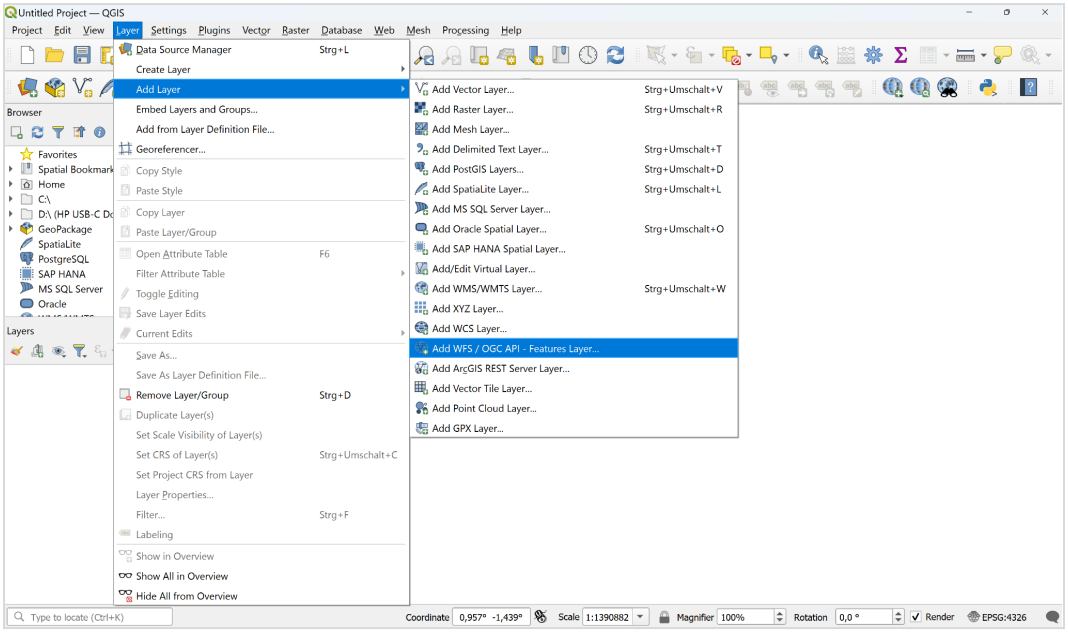

Fig. 12: QGIS - Add Layer - OGC API Features

3. In the dialog box, select "New" and enter the Name and URL of our NSG OGC API Features service.Type in the URL of the landing page of our service instance, which is http://localhost:5000/ and select the version of the feature service “OGC API Features”. In our case we limit the max. number of features to 250 since we just want to look up a sample of the dataset.

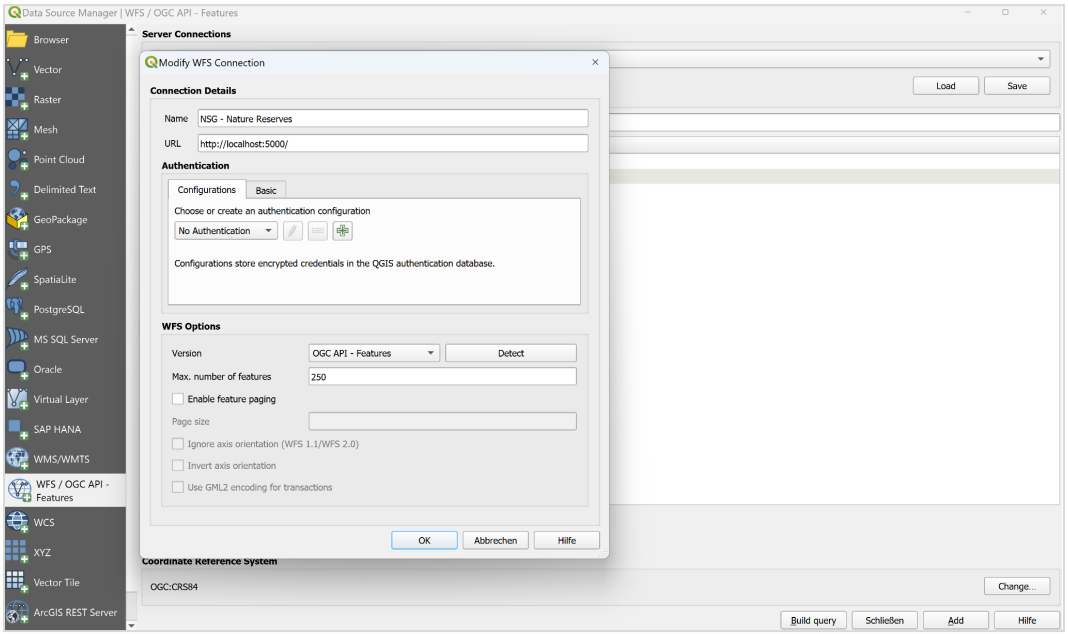

Fig. 13: QGIS - Connecting to the NSG OGC API Features service

4. Click on "Connect" to establish a connection to the service. Once connected, the available layers from the OGC Feature service will be displayed.

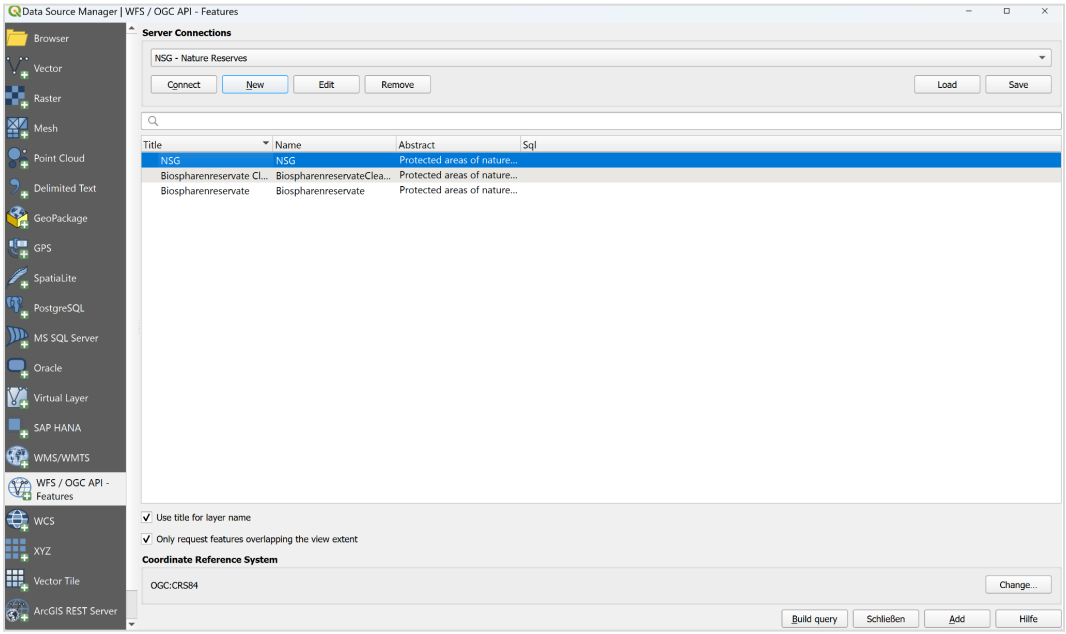

Fig. 14: QGIS - Selecting the NSG Feature Collection

5. Add the NSG Feature Collection as a new layer to your QGIS project.

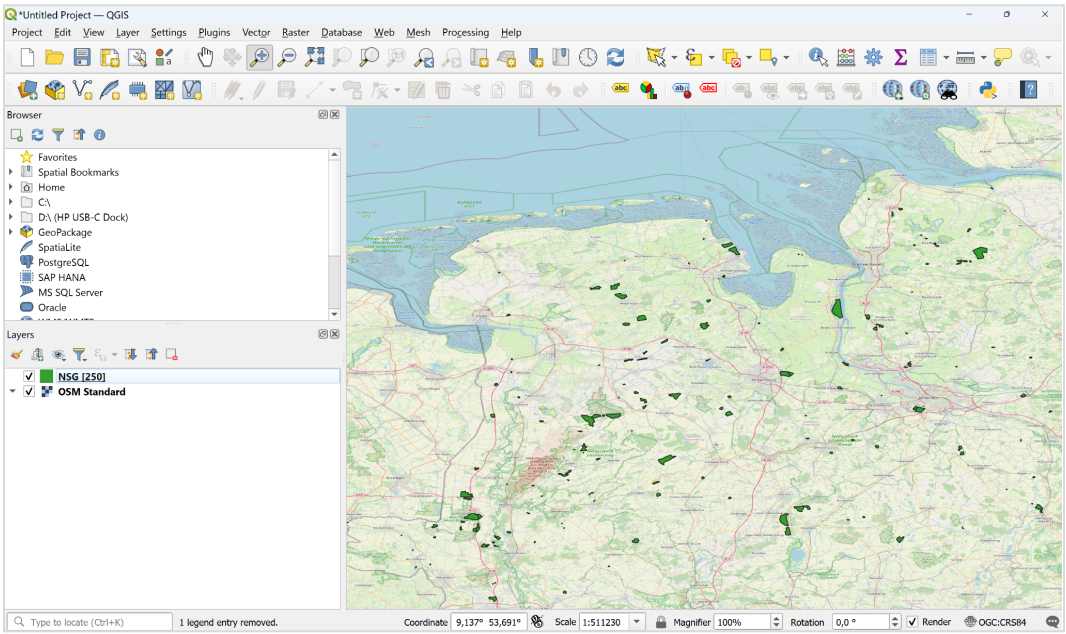

Fig. 15: Feature Collection added as a layer to the QGIS project

6. Now you can work with the layer as with any local resource, change styling, build subsets and use the data for data analysis.

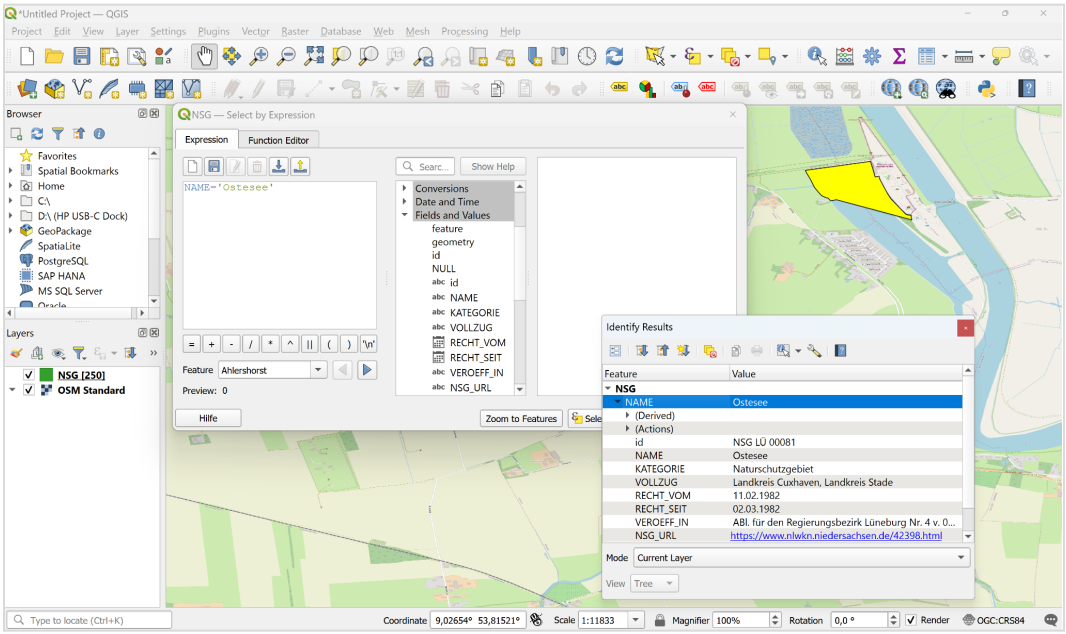

Fig. 16: Analyzing the OGC API based NSG Layer

The following video gives a short live demo of the described workflow.
[Video (Youtube): QGIS Access to Feature Data via the OGC API Features Interface]

3.5 Using Python to interact with the OGC-API Features service

In this section we will demonstrate how to use the Python scripting language for accessing the NSG data from our OGC API Feature service. You will learn how to interact with the service based on request and response messages that will be sent back and forth. This can be used to automate any communication workflow with the server which is important for building custom software applications.

We will use a Python Jupyter Notebook which is an executable document that contains text and code cells. You can go through the document and run the code step by step while the output of each step is displayed in the document. You can alter the code and re-run the document to explore the effects. A nice way of learning-by-doing..

After conducting the workflow of section 3.2 the Jupyter server and the notebook are already up-and-running. For accessing the server use the URL with the valid token that you can take from the logs of the juyter_container in the command window as explained in section 3.2.3.

Alternatively you can access the server via http://localhost:8888 from your browser. You will be asked for a valid token which is the one that you find in the logs of the jupyter_container as explained before.

Please start the notebook: “access_and_visualize_feature_data_NSG.jpynb”

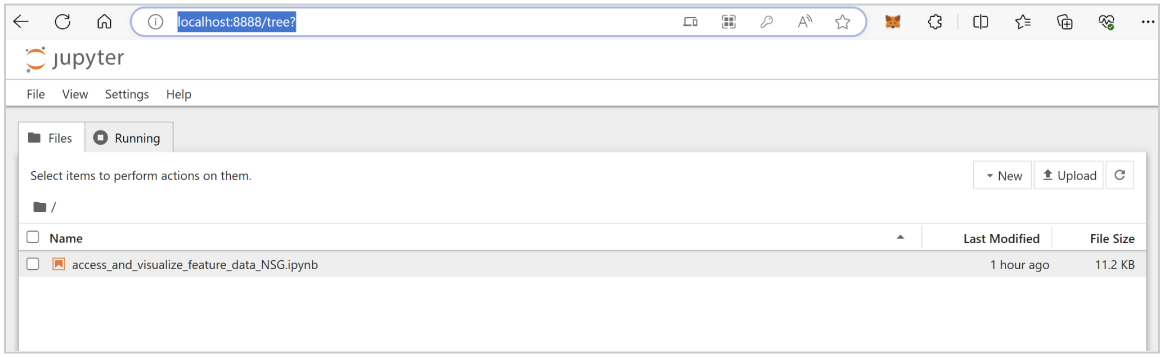

Fig. 17: Accessing the Python Notebook of this exercise

Now you should see the top of the notebook document that consists of cells that contain either markdown, code or the raw output of executed code.

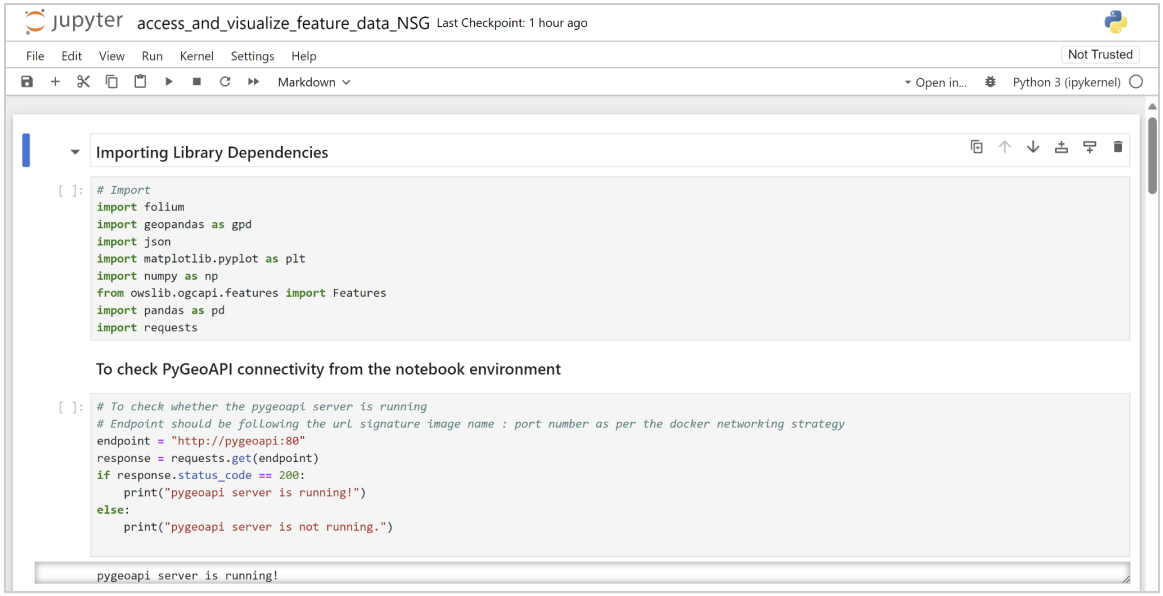

Fig. 18: Start working with the Jupyter Notebook

Start executing all cells one by one, by clicking the Play button, beginning with the topmost cell. While markdown cells will just be skipped, the code in the code cells will be executed which may take some seconds. So please wait for the result before you continue executing the next cells.

The Notebook will lead you through the following workflow:

- connecting to the OGC API Features service
- listing the feature collections that are offered by the service
- accessing the landing landing page of the NSG feature collection
- selecting feature data by their spatial extent
- creating buffers for selected features
- visualizing the feature data in an interactive map

The following video gives a short live demo of this workflow.

The following video gives a short live demo of the described workflow.
Video (Youtube): Python Access to Feature Data via the OGC API Features Interface

Now you can play around with the code to better understand the interactions and trying to implement changes following your own ideas.

For example you could try to query features by attributes and create your own map that presents the selected features in a certain symbolization. Or you could access the feature collection with biosphere reserves and explore that data source.

4. Discussion and Wrap up

In this tutorial, we learned how to access data provided via the OGC API Features interface.

Our NSG data service supports the Conformance Class HTML, which means that much of the content is also provided as HTML content. This offers the awesome opportunity to make the content accessible in a very user-friendly way, without requiring special software to use it. In addition, the service can now be indexed by search engines, so you can also search for data using Google and Bing and come across the services directly. Give it a try (e.g., search for "Gebäudedaten NRW, OGC API features").

So far, there are still very few operational service instances, but this will change drastically in the near future.

Furthermore, we have seen that the same service can also be used via standard software such as QGIS or via programming languages such as Python.For machine-to-machine communication, these systems do not use HTML but the JSON or GeoJSON encodings that are offered by the service as an alternative for all available resources.

Now you can use the server on your local computer to play around with the service from the browser, QGIS and Python. Though we did not focus on how to set up a service instance you might even look up the installation in its details and learn from it.

The following resources may help you to further dive into the topic and to deepen your understanding:

- [1] Github repository of this tutorial: https://github.com/oer4sdi/OER-DataAccessVia-OGC-API-Features
- [2] OGC Specification of the OGC API Features interface: https://ogcapi.ogc.org/features/
- [3] An Introduction to OGC API - Features — OGC e-Learning 2.0.0 documentation: http://opengeospatial.github.io/e-learning/ogcapi-features/text/basic-main.html
- [4] PyGeoAPI documentation https://docs.pygeoapi.io/en/stable/

- [5] Documentation on the OWSLIB library that you can use with Python to work with OGC Web Services
  https://owslib.readthedocs.io/en/latest/

If you face any problems with this tutorial or if you want to contribute your ideas on how to improve the material you are invited to add issues to our GitHub repository (https://github.com/oer4sdi/OER-DataAccessVia-OGC-API-Features).