NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Automatic annotation of Earth Observation data

Panagiota C. Papadima
Marianthi E. Theologi

Supervisors: **Manolis Koubarakis,** Professor
Despina Athanasia Pantazi, PhD Candidate

ATHENS

SEPTEMBER 2021

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αυτόματη επισημείωση δεδομένων παρατήρησης της γης

Παναγιώτα Χ. Παπαδήμα
Μαριάνθη Ε. Θεολογή

Επιβλέποντες: **Μανόλης Κουμπαράκης,** Καθηγητής
Δέσποινα Αθανασία Πανταζή, Υποψήφια Διδάκτωρ

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2021

BSc THESIS

Automatic annotation of Earth Observation data

Panagiota C. Papadima
S.N.: 1115201400143
Marianthi E. Theologi
S.N.: 1115201400047

SUPERVISORS: **Manolis Koubarakis,** Professor
Despina Athanasia Pantazi, PhD Candidate

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αυτόματη επισημείωση δεδομένων παρατήρησης της γης

Παναγιώτα Χ. Παπαδήμα
Α.Μ.: 1115201400143
Μαριάνθη Ε. Θεολογή
Α.Μ.: 1115201400047

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Μανόλης Κουμπαράκης,** Καθηγητής
Δέσποινα Αθανασία Πανταζή, Υποψήφια Διδάκτωρ

ABSTRACT

The current thesis presents how we can automatically annotate Earth Observation datasets that include Sentinel products offered by Copernicus based on the OGC 17-003r2 Earth Observation Dataset Metadata GeoJSON(-LD) Encoding Standard. To automate the production of JSON-LD annotations, we constructed an RDF graph with triplets of the data we collected from the Copernicus Open Access Hub. Finally, we have implemented an API to provide access to the automation procedure and a User Interface which consumes this API.

SUBJECT AREA: Semantic Web

KEYWORDS: Earth Observation, Data annotation, Schema.org, metadata, Python, JSON, Linked data, API, User Interface

ΠΕΡΙΛΗΨΗ

Στην παρούσα πτυχιακή εργασία παρουσιάζουμε την αυτοματοποίηση της κωδικοποίσης τηλεσκοπικών συνόλων δεδομένων από δορυφορικά προϊόντα, που μας παρέχονται από το πρόγραμμα Copernicus, συμφωνα με το επίσημο λεξιλόγιο OGC 17-003r2. Για την αυτοματοποίηση της διαδικασσίας παραγωγής ενός JSON-LD αρχείου, κατασκευάσαμε έναν RDF γράφο με τις τριπλέτες από τα δεδομένα που συλλέξαμε από τους δορυφόρους του προγράμματος Copernicus. Τέλος, υλοποιήσαμε ένα API με το οποίο οι χρήστες μπορούν να έχουν πρόσβαση στην αυτοματοποίση και ένα User Interface με το οποίο το API γίνεται πιο φιλικό προς τον χρήστη.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: σημασιολογικός ιστός

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: τηλεσκόπιση, επισημείωση δεδομένων, διασυνδεδεμένα δεδομένα, μεταδεδομένα, σύνολα δεδομένων, εφαρμογή

To our families

ACKNOWLEDGEMENTS

Για τη διεκπεραίωση της παρούσας Πτυχιακής Εργασίας, θα θέλαμε να ευχαριστήσουμε τους επιβλέποντες, καθηγητή Μανόλη Κουμπαράκη και την υποψήφια διδάκτωρ Δέσποινα-Αθανασία Πανταζή για τη συνεργασία και την πολύτιμη συμβολή τους στην ολοκλήρωση της.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

1. INTRODUCTION

Earth Observation(EO) is one of the most critical assets of humanity, as it allows us to gather information about planet Earth's physical, chemical and biological systems. There are many different kinds of Earth observations, such as a bird-watcher's notes, radar and sonar images, analyses of water or soil samples and others. These observations produce EO datasets, which are used everyday by various applications that contribute to humanity's evolution. Some of the most important applications of Earth Observation can be met among forecasting weather, tracking biodiversity and wildlife trends, measuring land-use change and monitoring and responding to physical disasters.

Most of these datasets exist in the archives of space or earth programs and agencies, and can only be found through complex user interfaces. Easier access to this data, would allow more people to make use of it and thus help in accomplishing more scientific achievements. It is important to make this information available on the Web in the form of linked data. In this way it will be easier for developers, who are not experts in Earth Observation, to access this data and benefit from it.

Through the last years, search engine technologies have made a huge progress on the results they can provide to users' questions. By introducing semantic technologies, search engines now optimize their results by understanding the context of the search, rather than ranking the keywords' appearance frequency. Although the technologies exist and we can benefit from them for most of the resources, little progress has been made on the ability to query and find datasets.

The goal of this thesis, is to automate the process of annotating EO datasets to make them available on the Web as linked geospatial data. This will make Earth Observation datasets discoverable by search engines like Google or Yahoo and will provide easier access to them for the developer community. An API was implemented to provide access to the automation process, which can be used by other developers. Furthermore, we created a graphical User Interface(UI) which consumes this API. The full source of the implementation can be found in the following github repository: `https://github.com/nypen/insight`

The thesis has the following organisation: in Chapter 2 we introduce the concepts of the Semantic Web, the linked data and JSON-LD, the OGC 17003r2 vocabulary and the extension of Schema.org based on this vocabulary. In Chapter 3, we describe our methodology and research steps we followed to study an EO dataset and how we find it, the OGC 17-003r2 vocabulary with the classes and their properties that we used, the annotation procedure we did based on the previous vocabulary and the RDF graph construction which leads to the JSON-LD production. Chapter 4 explains the implementation steps for the API and the User Interface by delving in the technologies and libraries we chose to use and finally a description of how our interface works. Last but not least, in Chapter 5 we summarize our contributions and discuss future work.

2. RELATED WORK

In this chapter we give a first understanding of the specific fields of the Semantic Web, Linked Data and the Schema.org vocabulary. We also introduce JSON-LD [Section 2.2], OGC 17-003r2[1] and the research done at the Thesis *"Encoding and Validation of Earth Observation Metadata using Schema.org and SHACL"* [5] upon which we based our procedure to annotate satellite images.

2.1 Semantic Web

The Semantic Web, as first expressed by Tim Berners-Lee in 1989 [2], is the extension of the current World Wide Web. Day by day, we post content online that can primarily be consumed by humans but not by machines. For example, we can browse our photo gallery or our calendar online, but we do not have the ability to combine this information and go back to what we were doing on a particular day that we shot a particular photo. Though all the data is available online, machines cannot process it easily because of the missing links. The goal of the Semantic Web is to provide a framework, through which data is allowed to be shared and reused across application, enterprise, and community boundaries.

Semantics, or meaning, of information on the Web is formally defined by ontologies. To convey the meaning of information, according to the Semantic Web, sets of triples are used. Those triples consist of a subject, a predicate and an object rather like the subject, verb and object in a sentence. The Semantic Web is being built and standardized based on the W3C standards: Resource Description Framework (RDF), SPARQL Protocol and RDF Query Language (SPARQL)[2], Ontology Web Language (OWL)[3], Extensible Markup Language (XML)[4], and Uniform Resource Identifier (URI)[5].

RDF is a standard for describing web resources and data interchange. The way RDF connects data is via triples. Each triple has three parts:

1. a subject,
2. an object, and
3. a predicate (also called a property) that denotes a relationship.

Subjects and objects are identified by URIs.

The triple can be presented by a node and directed-arc diagram. In this diagram each triple is represented as a node-arc-node link and shows the relationship between the things indicated by the nodes that it connects. A set of such triples is called an *RDF graph* and consists a web of information about related things.

These nodes of the RDF graph are URI references, blank nodes or literals. In RDFLib[6],

[1] http://docs.opengeospatial.org/is/17-003r2/17-003r2.html
[2] https://www.w3.org/TR/sparql11-query/
[3] https://www.w3.org/OWL/
[4] http://www.xml.org
[5] https://en.wikipedia.org/wiki/Uniform_Resource_Identifier
[6] https://github.com/RDFLib/rdflib

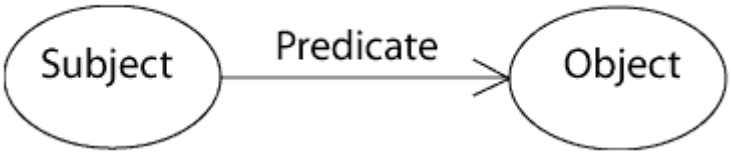

Figure 1: RDF graph node

these node types are represented by the classes URIRef, BNode, and Literal as explained below:

- A BNode is a node where the exact URI is not known.
- A URIRef is a node where the exact URI is known. URIRefs are also used to represent the properties/predicates in the RDF graph.
- Literals represent attribute values, such as a name, a date, a number, etc. The most common literal values are XML data types, e.g. string, int, etc.

2.2 Linked Data and JSON-LD

Linked Data[7] helps the development of Web applications and empowers the people who use and circulate information on the Web. It is a way to create a network of standards-based, machine-readable data across Web sites. It permits an application to start at one piece of Linked Data, and follow ingrained links to other pieces of Linked Data from different sites across the internet.

JavaScript Object Notation (JSON)[8] is becoming the most popular data interchange format in Web-based applications. A set of objects characterized by name/value pairs is what JSON provides to web developers and helps the need to annotate metadata found on the web.

JSON-LD (JavaScript Object Notation for Linking Data)[9], as mentioned in its documentation, is a lightweight Linked Data format. It is based on the JSON format and provides a way to help JSON data interact at Web-scale. JSON-LD is a data format commonly used for programming environments, REST Web services, and unstructured databases such as Apache CouchDB[10] and MongoDB[11].

The expression of Linked Data in JSON is not the only procedure that JSON-LD simplifies. With JSON-LD developers can easily add semantics to already existing JSON documents. The need of developers to transform JSON to a more semantically rich format lead to this entity-centric approach. JSON-LD creates a description of the data in the form of a context which links objects and their properties in a JSON document. Finally, the format is intended to be fast to parse and generate, compatible with document processing, and has a small memory requirement for it to run.

[7] https://www.w3.org/standards/semanticweb/data
[8] https://www.json.org/json-en.html
[9] https://json-ld.org
[10] http://couchdb.apache.org
[11] https://www.mongodb.com

2.3 OGC 17-003r2

For automating the annotation of metadata collected from Copernicus satellites, a formation of them into JSON-LD encoding is needed.

The Open Geospatial Consortium 17-003r2[12] standard contains a GeoJSON and JSON-LD encoding for Earth Observation (EO) metadata for datasets. As described in its documentation, OGC 17-003r2 can be used for the encoding of more complex and specialized metadata based on OGC 10-157r4 which is the Earth Observation Metadata Profile of Observations and Measurements (O&M) and the Unified Metadata Model for Granules (UMM-G) conceptual model.

As mentioned in OGC 17-003r2 the GeoJSON encoding is defined as a compaction through a normative context of the proposed JSON-LD encoding, with some extensions. Therefore, the JSON-LD encoding can also be applied to other RDF encodings including RDF XML and RDF Turtle.

According to OGC 17-003r2, the main class of EO Datasets is *EarthObservation*, which is connected to the class *Properties*. Class *Properties* has the object property *properties* and it is connected to four other classes: *AcquisitionInformation, Link, Offering,* and *ProductInformation*. The classes and their properties are further explained below:

1. *AcquisitionInformation* class is connected to the following classes:
   (a) *AcquisitionParameters* class includes all the data properties and object properties of the OGC 17-003r2 class *AcquisitionParameters*: *type, acquisitionSubType, startTimeFromAscendingNode(ascendingNodeDate), completionTimeFromAscendingNode(ascendingNodeDate), orbitNumber, wrsLongitudeGrid, wrsLatitudeGrid, tileId, groundTrackUncertainty, cycleNumber, acquisitionStation acquisitionAngles, waveLengths, aquisitionType, antennaLookDirection*
   For *acquisitionAngles* there is the class *eoAcquisitionAngles*, includes all the properties of the OGC 17-003r2: *type, illuminationAzimuthAngle, illuminationZenithAngle, illuminationElevationAngle, incidenceAngle, minimumIncidenceAngle, maximumIncidenceAngle, incidenceAngleVariation, acrossTrackIncidenceAngle, alongTrackIncidenceAngle, instrumentAzimuthAngle, instrumentZenithAngle, instrumentElevationAngle, pitch, roll, yaw*
   For *waveLengths* there is the property *WavelengthInformation* and it includes all the properties of the OGC 17-003r2 class *WavelengthInformation*: *type, discreteWavelengths, startWavelength, endWavelength, spectralRange, wavelengthResolution*
   (b) *Instrument* class includes the properties of the OGC 17-003r2 class *Instrument*: *type, id, sensorType, instrumentShortName, description*
   (c) *Platform* class includes the properties of the OGC 17-003r2: *type, id, platformShortName, platformSerialIdentifier, orbitType*
2. *Link* class information includes: *type, alternates, via, previews, data, qualityReport, related, up*
3. *Offering* class information includes the properties of the OGC 17-003r2: *operations, contents, styles, extension*

[12] https://docs.opengeospatial.org/is/17-003r2/17-003r2.html#table_14

4. *ProductInformation* class information includes the properties of the OGC 17-003r2: *type, productType, size, statusSubType, statusDetail, availabilityTime, timeliness, productGroupId, productVersion, archivingCenter, archivingDate, referenceSystemIdentifier, compositeType, format, processingMethod, processingMethodVersion, processingCenter, processingDate, processingLevel, processingMode, processorName, processorVersion, productContentsType, cloudCover, snowCover, qualityInformation*

2.4 Extending Schema.org based on OGC 17-003r2

Schema.org is a vocabulary created by the large search engines Google, Microsoft, Pinterest, Bing, Yandex and others with the goal to create, provide and develop schemas for structured data on the web.

To be able to use and benefit from Schema.org initiative developers can use a variety of encodings such as JSON-LD , RDFa (Resource Description Framework in Attributes)[13] or Microdata[14], which was created to make RDFa more simple. These syntaxes cover objects, relationships between objects and actions and give the opportunity to webmasters to extend them if needed.

Schema.org contains a core vocabulary which describes the most basic and essential entities webmasters use in their jobs. However, for the annotation of geospatial metadata that we get from Satellite images, the vocabularies we use need to contain deeper and more specialized entities. That is the reason of the creation of extensions which are provided by a well-documented extension model. Such an extension was introduced by the diploma thesis [5] for the Earth observation dataset metadata Vocabulary, as described in OGC 17-003r2.

This extension proposes to not include the *Properties* class, as described in OGC 17-003r2, and to include the properties of the four classes linked to it (*AcquisitionInformation, Link, Offering,* and *ProductInformation*) in the new Schema.org class EarthObservation. The classes and properties according to this extension are:

1. *AcquisitionInformation* class includes the following properties:

   - *eoAcquisitionParameters* property expressed with Schema.org class *AcquisitionParameters* and includes the data and object properties of OGC 17-003r2 class *AcquisitionParameters*

     For the object property *acquisitionAngles*, there is a new property, the *eoAcquisitionAngles* property. It is expressed with Schema.org class *AcquisitionAngles* and includes all the properties of the OGC 17-003r2 class *AcquisitionAngles*
   - *eoInstrument* property expressed with Schema.org class *Instrument* and includes the data and object properties of OGC 17-003r2 class *Instrument*
   - *eoPlatform* property expressed with Schema.org class *Platform* and includes the data and object properties of OGC 17-003r2 class *Platform*

[13] `https://rdfa.info/`

[14] `https://html.spec.whatwg.org/multipage/microdata.html`

2. *Link* class information is defined OGC 17-003r2 class, as the properties it includes can be represented by the url properties that are already defined in *Thing* and *CreativeWork* Schema.org classes.

3. *Offering* class information is defined as the Schema.org property *eoOffering* and is expressed with class *Offering* and includes the properties of the OGC 17-003r2 class *Offering*

4. ProductInformation class information is defined as the Schema.org property *eoProductInformation*, which has as its expected type the Schema.org class *Product* and includes the properties of the OGC 17-003 class *ProductInformation*

2.5 Summary

In this chapter we discuss about the specific fields of the Semantic Web, Linked Data and the Schema.org vocabulary. Moreover we introduce JSON-LD, OGC 17-003r2 and the research presented in diploma thesis [5].

3. METHODOLOGY

In this chapter, we describe the process we followed to annotate the metadata of an Earth Observation dataset as described in OGC-17003r2 standard and construct a generalized graph to produce a final JSON-LD encoding.

3.1 Dataset

In order to apply our automated annotation of Earth Observation metadata, we used satellite images of Sentinel-1[1] (Level-0 or Level-1 user products), Sentinel-2[2] (Level-1C and Level-2A user products), Sentinel-3[3] (Level-1 and Level-2 Land user products for the OLCI, SLSTR and SRAL instruments) and Sentinel-5P[4] (Level-1B and Level-2 user products for the TROPOMI instrument) missions, offered by the Copernicus Open Access Hub[5] (OAH), which provides completely free and open access to its products via the Data Hub Interface or via requests for products on API Hub[6].

The API Hub follows the OData protocol, which defines a set of practices for building RESTful APIs in a generic way. Each resource is identified by a Uniform Resource Identifiers (URIs). URIs used by an OData service have up to three significant parts:

- the *Service Root URI*, which identifies the root of the OData service. The available OData Service Root URI for the API Hub are the following:

  https://scihub.copernicus.eu/apihub/odata/v1/

  https://s5phub.copernicus.eu/dhus/odata/v1/

- the *Resource Path* which identifies the resource collection entity to be interacted with. Copernicus, as far, provides access to the following resources: /Products, /DeletedProducts, /Collections, /Products('Id), /DeletedProducts('Id'), /Collections('Id').

- the system *Query Options* part which refines the results. For example, "format=json"

The OData URI addressing the resource "/Products('Id')" provides access to the data files stored in the Data Hub archive for each individual Sentinel product. Each entry includes:

- the data file name,
- the Id corresponding to the Universally Unique Identifier (UUID),
- the download URI "Products('Id')/$value",
- the URIs of the navigation properties "/Nodes" and "/Attributes" and
- the properties of the individual entity.

[1] `https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-1`
[2] `https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-2`
[3] `https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-3`
[4] `https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-5p`
[5] `https://scihub.copernicus.eu/dhus/#/home`
[6] `https://scihub.copernicus.eu/twiki/do/view/SciHubWebPortal/APIHubDescription`

The values we use to annotate a sentinel product dataset are requested from the following URIs:

$$\underbrace{https://apihub.copernicus.eu/apihub/odata/v1}_{\text{Service Root URI}}\underbrace{/Products(\text{'id'})/Attributes}_{\text{Resource Path}}\underbrace{\$format=json}_{\text{Query Options}}$$

$$\underbrace{https://s5phub.copernicus.eu/dhus/odata/v1/}_{\text{Service Root URI}}\underbrace{Products(\text{'id'})/Attributes}_{\text{Resource Path}}\underbrace{\$format=json}_{\text{Query Options}}$$

E.g. `https://apihub.copernicus.eu/apihub/odata/v1/Products('a239b5b3-62a4-4509-9bd6-ff6db6a3b40b')/Attributes?$format=json`

A Copernicus Open Access API response contains the set of the user product's attributes. Each attribute is described by an object as shown in Figure 2. Every distinct property of a sentinel product is identified among the rest by the property "Id". To simplify the annotation process, for each property we collect only the *Id* and *Value* properties. The rest of the properties (*_metadata, Name, ContentType, ContentLength, Category*) do not offer any valuable information for the annotation process, thus there is no need to store them.


```
{
  "__metadata":{
    "id":"https://apihub.copernicus.eu/apihub/odata/v1/Products('2b17b57d-fff4-4645-b539-91f305c27c69')/Attributes('Sensing%20start')",
    "uri":"https://apihub.copernicus.eu/apihub/odata/v1/Products('2b17b57d-fff4-4645-b539-91f305c27c69')/Attributes('Sensing%20start')",
    "type":"DHuS.Attribute"
  },
  "Id":"Sensing start",
  "Name":"Sensing start",
  "ContentType":"text/plain",
  "ContentLength":"24",
  "Value":"2016-01-17T10:34:51.922Z",
  "Category":"product"
},
```


Figure 2: A Copernicus Response Object Example

There are attributes retrieved from Copernicus, that are mission and product type specific, like "Cloud Cover Percentage (%)" of Sentinel 3 (SY-2-SYN, SY-2-V10, SY-2-VG1, SY-2-VGP, SY-2-AOD, OL-2-LFR, OL-2-LRR product types) or "Cloud cover percentage" of Sentinel 2 (S2MSI2Ap, S2MSI2A, S2MSI1C product types) , while there are others that are provided by all or most of the sentinel products. Properties such as "Ingestion date", "Creation Date", "Generation Date", "Title", "Identifier" and other are provided for all Sentinel products despite product type.

Other attributes may represent the same value but can be met with different names. For example:

- *Instrument short name* of Sentinel-5P's products and *Instrument abbreviation* of the rest sentinel products.
- *Processing baseline* of Sentinel-2 products and *Baseline Collection* property of Sentinel-3 products.
- *Snow ice percentage* of Sentinel-2 products (S2MSI2Ap, S2MSI2) and *Snow or Ice Cover Percentage (%)* of most of Sentinel-3 products (AOL-2-LFR, OL-2-LRR,SY-2-SYN, SY-2-V10, SY-2-VG1, SY-2-VGP, SY-2-AOD)

3.2 OGC 17-003r2 Vocabulary

OGC 17-003r2 describes a GeoJSON-based serialization syntax, as described in Section 2.3, which consists a set of classes that represent geometries or features. For the data annotation we comply with the objects and their properties that are described in the following tables(1, 2, 3, 4, 5, 6, 7, 8, 9 and 10) as in the OGC 17-003r2.

According to the OGC 17-003r2, the main class is *EarthObservation* which is connected to the class *Properties*. Class *Properties* is connected to the classes *Link*, *Offerings*, *AcquisitionInformation* and *ProductInformation*. Graphically the Earth Observation Dataset can be described like in Figure 3.

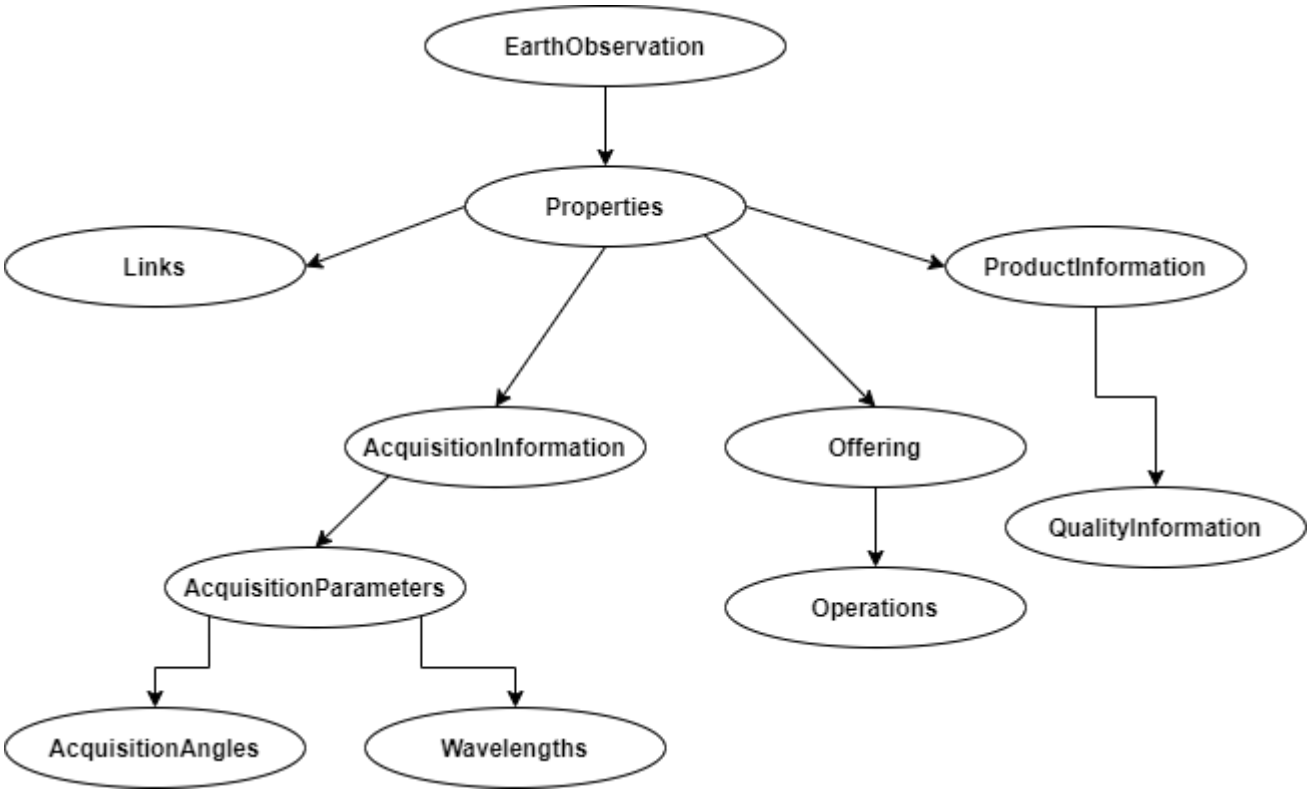

Figure 3: Properties used in OGC-17-003r2

Each class includes its own properties according to the OGC 17-003r2. In the next tables we show the properties each class contains:

Table 1: Object properties of the class Properties

JSON Property	Expected Type
type	String
links	Links
offerings	Array of Offering
status	String
acquisitionInformation	Array of AcquisitionInformation
productInformation	ProductInformation

Table 2: Object properties of the class Link

JSON Property	Expected Type
href	String
type	String
title	String
length	Integer
category	String
expression	String
conformsTo	String

Table 3: Object properties of the class AcquisitionInformation

JSON Property	Expected Type
platform	**Platform** [Table 6]
instrument	**Instrument** [Table 7]
acquisitionParameters	**AcquisitionParameters** [Table 5]

Table 4: Object properties of the class ProductInformation

JSON Property	Expected Type
type	String
productType	String
size	Integer
statusSubType	String
statusDetail	String
availabilityTime	DateTime
timeliness	String
productGroupId	String
productVersion	String
archivingCenter	String
archivingDate	DateTime
referenceSystemIdentifier	String (URI)
qualityInformation	**QualityInformation** [Table 8]

Table 5: Object properties of the class AcquisitionParameters

JSON Property	Expected Type
type	String
acquisitionType	String
acquisitionSubType	String
startTimeFromAscendingNode	Integer
completionTimeFromAscendingNode	Integer
relativeOrbitNumber	Integer
wrsLongitudeGrid	String
wrsLatitudeGrid	String
tileId	String
groundTrackUncertainty	Double
cycleNumber	Integer
antennaLookDirection	String
acquisitionAngles	**AcquisitionAngles** [Table 9]
operationalMode	String
swathIdentifier	String
polarisationMode	String (S, D, T, Q, UNDEFINED)
polarisationChannels	String
resolution	String
verticalResolution	Double
wavelengths	**WavelengthInformation** [Table 10]
measurementType	String
dopplerFrequency	Double
samplingRates	Array of Double

Table 6: Object properties of the class Platform

JSON Property	Expected Type
type	String
id	String (URI)
platformShortName	String
platformSerialIdentifier	String
orbitType	String

Table 7: Object properties of the class Instrument

JSON Property	Expected Type
type	String
id	String (URI)
sensorType	String
instrumentShortName	String
description	String

Table 8: Object properties of the class QualityInformation

JSON Property	Expected Type
type	String
qualityStatus	String
qualityDegradation	Double
qualityDegradationQuotationMode	String
qualityDegradationTag	String

Table 9: Object properties of the class AcquisitionAngles

JSON Property	Expected Type
type	String
illuminationAzimuthAngle	Double
illuminationZenithAngle	Double
illuminationElevationAngle	Double
incidenceAngle	Double
minimumIncidenceAngle	Double
maximumIncidenceAngle	Double
incidenceAngleVariation	Double
acrossTrackIncidenceAngle	Double
alongTrackIncidenceAngle	Double
instrumentAzimuthAngle	Double
instrumentZenithAngle	Double
instrumentElevationAngle	Double
pitch	Double
roll	Double
yaw	Double

Table 10: Object properties of the class WavelengthInformation

JSON Property	Expected Type
type	String
discreteWavelengths	Array of Double
startWavelength	Double
endWavelength	Double
spectralRange	String
wavelengthResolution	Double

3.3 Annotation

To annotate a sentinel product's properties to the properties of OGC-17003r2 we used information published by Copernicus official Website[7] to find what each property represents and map it to its equivalent OGC-17003r2 property respecting their accepted types and values.

For example in Figure 4 we see the response from Copernicus website for the attribute with *"Id"* : *"Acquisition Type"*


```
{
  "__metadata":{
    "id":"https://apihub.copernicus.eu/apihub/odata/v1/Products('2b17b57d-fff4-4645-b539-91f305c27c69')/Attributes('Acquisition%20Type')",
    "uri":"https://apihub.copernicus.eu/apihub/odata/v1/Products('2b17b57d-fff4-4645-b539-91f305c27c69')/Attributes('Acquisition%20Type')",
    "type":"DHuS.Attribute"
  },
  "Id":"Acquisition Type",
  "Name":"Acquisition Type",
  "ContentType":"text/plain",
  "ContentLength":"7",
  "Value":"NOMINAL",
  "Category":"product"
},
```


Figure 4: Acquisition Type from the Copernicus website

Based on the property's value, type and meaning it should be mapped to OGC-17-003r2's property *"properties.acquisitionInformation.acquisitionParameters.acquisitionType"* property (Figure 5)

acquisitionType $..acquisitionParameters.acquisitionType	Used to distinguish at a high level the appropriateness of the acquisition for "general" use, whether the product is a nominal acquisition, special calibration product or other. Values: - NOMINAL - CALIBRATION - OTHER	Domain: AcquisitionParameters Range: String	One (mandatory)

Figure 5: OGC 17-003r2 Acquisition Type

Tables **11** and **12** contain the properties we annotated following the process described above.

[7] https://sentinels.copernicus.eu/web/sentinel/home

Table 11: Dataset annotation

Schema.org Property	Sentinel value
properties.status	Status
properties.title	Identifier
properties.identifier	Identifier
properties.date	Date
properties.created	Ingestion Date
properties.available	Generation Date, Creation Date
properties.acquisitionInformation. acquisitionParameters.acquisitionType	Acquisition Type
[..].acquisitionParameters. acquisitionSubType	Mission datatake id
[..].acquisitionParameters. relativeOrbitNumber	Relative orbit (start)
[..].acquisitionParameters.tileId	Tile Identifier
[..].acquisitionParameters.cycleNumber	Cycle number
[..].acquisitionParameters.operationalMode	Instrument mode, Mode
[..].acquisitionParameters.swathIdentifier	Instrument swath
[..].acquisitionParameters. polarisationChannels	Polarisation
[..].acquisitionParameters.orbitDirection	Orbit Direction (start), Pass direction
[..].acquisitionParameters.lastOrbitDirection	Orbit Direction (stop), Orbit Direction (start), Pass direction
[..].acquisitionParameters.orbitDuration	PDU Duration (s)
[..].acquisitionParameters.orbitNumber	Orbit number (start)
[..].acquisitionParameters.lastOrbitNumber	Orbit number (stop), Orbit number (start)
[..].acquisitionParameters. beginningDateTime	Sensing start
[..].acquisitionParameters.endingDateTime	Sensing stop
properties.acquisitionInformation. acquisitionParameters.acquisitionAngles. illuminationAzimuthAngle	Illumination Azimuth Angle
[..].acquisitionAngles. illuminationZenithAngle	Illumination Zenith Angle
properties.acquisitionInformation.platform. platformShortName	Satellite name, Satellite
[..].platform.platformSerialIdentifier	Platform serial identifier
properties.acquisitionInformation. instrument.instrumentShortName	Instrument abbreviation, Instrument short name
[..].instrument.description	Instrument description text

Table 12: Dataset Annotation

Schema.org Property	Sentinel value
properties.productInformation.productType	Product type
properties.productInformation.size	Size
properties.productInformation. availabilityTime	Ingestion Date
properties.productInformation.timeliness	Timeliness Category
properties.productInformation. productGroupId	Product class
properties.productInformation. processingMethodVersion	Baseline Collection, Processing baseline
properties.productInformation. processingCenter	Processing Facility Name
properties.productInformation. processingLevel	Processing Level, Product level
properties.productInformation. processingMode	Processor mode abbreviation
properties.productInformation. processorVersion	Processor version
properties.productInformation.cloudCover	Cloud cover percentage, Cloud Cover Percentage (%)
properties.productInformation.snowCover	Snow ice percentage, Snow or Ice Cover Percentage (%)

3.4 RDF Construction

To automate the process of producing the final JSON-LD document we need to costruct a generalized graph. Generalized RDF triples, graphs, and datasets differ from normative RDF triples, graphs, and datasets only by allowing IRIs, blank nodes and literals to appear in any position, i.e., as subject, predicate, object or graph names.

Each triplet of the graph is formed by three components: A *subject*, a *predicate* and an *Object*. All of them can be any of:

- A *BNode*, where the exact URI is not known
- A *URIRef*, where the exact URI is known. URIRefs are also used to represent the properties
- A *Literal* to represent attribute values, such as a name, a data a number, etc. The most common literal values are XML data types (e.g. string, int).

In the following sections we describe how we costructed this graph, explaining in detail the process for the most complex nodes and visualizing the final result.

3.4.1 The *earthObservation* node

The root node of the graph, where all the properties of an Earth Observation dataset are linked to, is a URIREF node as long as the id is known. Each property of the root node is expressed through triplets where the subject is the URIREF of the Earth Observation, the predicate is the property and the object is either a blank node or a URIREF. Since eoProperties and geometry properties are not identified with a URI (id), blank nodes are used for the connection.

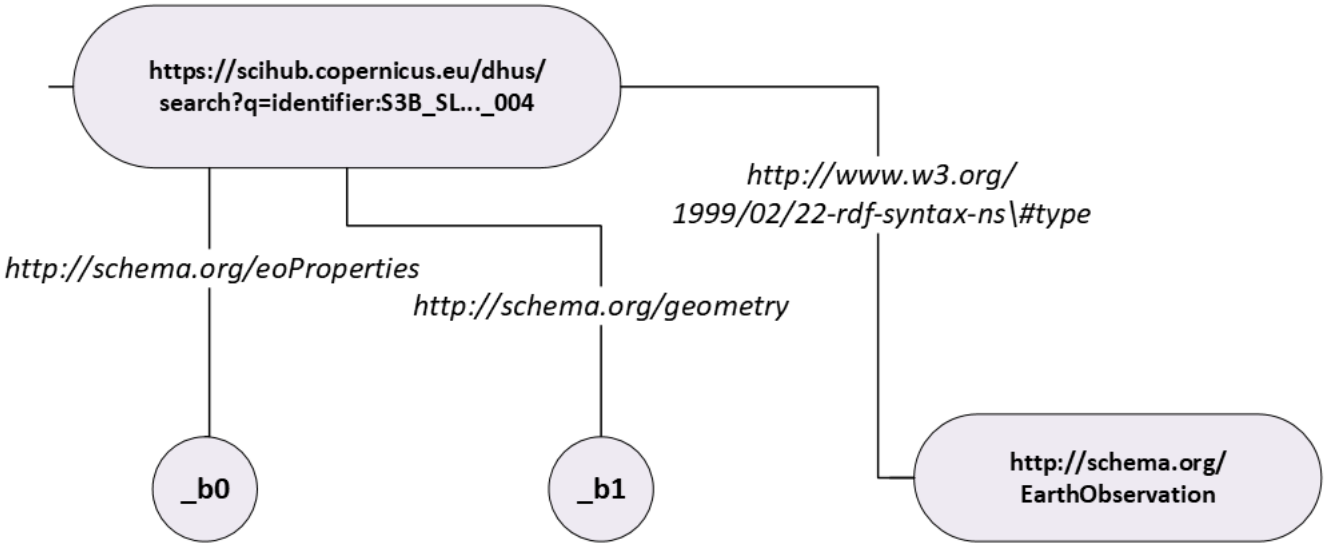

Figure 6: Earth Observation root node

3.4.2 The *properties* node

The ”eoProperties” node (_b0) of the graph, is a blank node linked to its properties in a similar way.

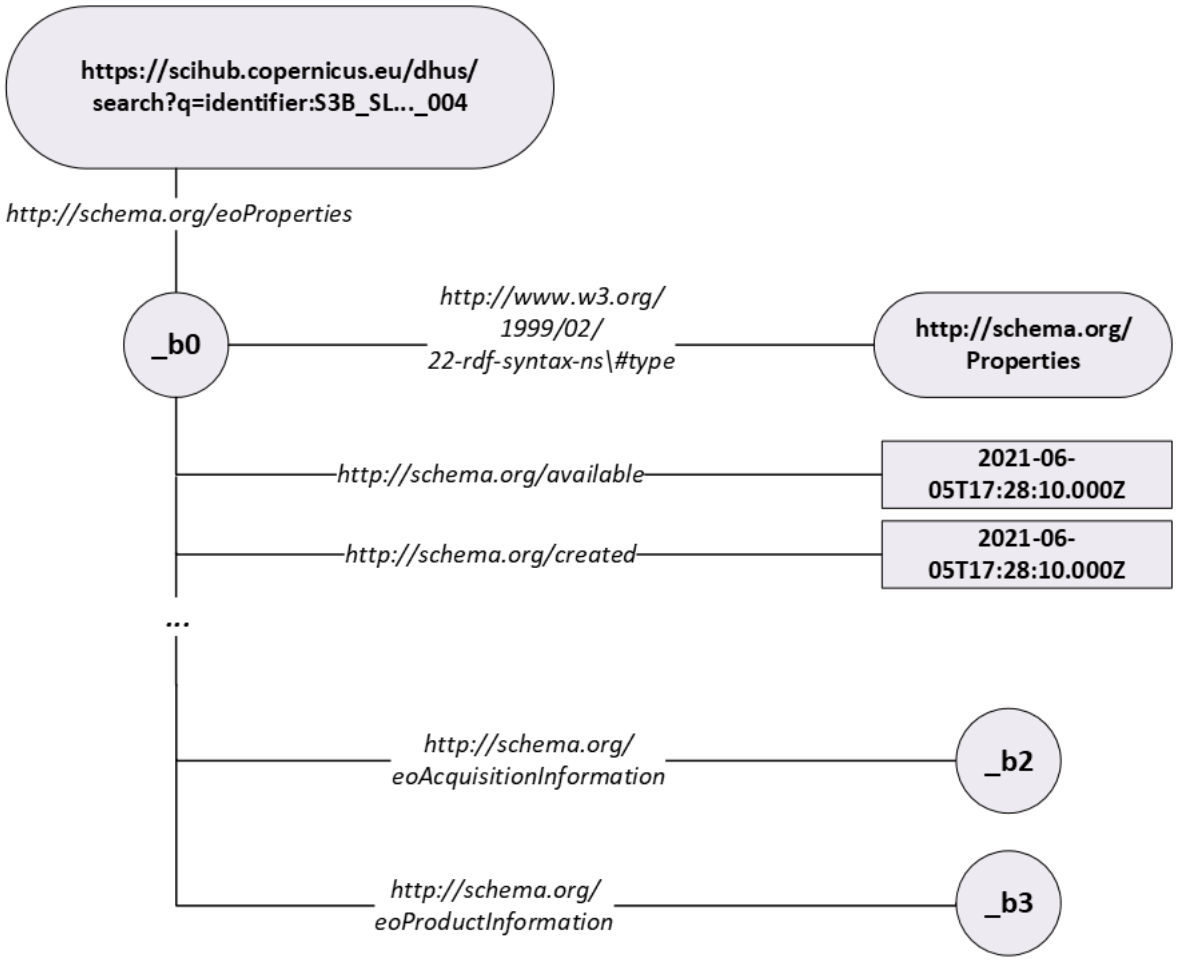

Figure 7: EoProperties node

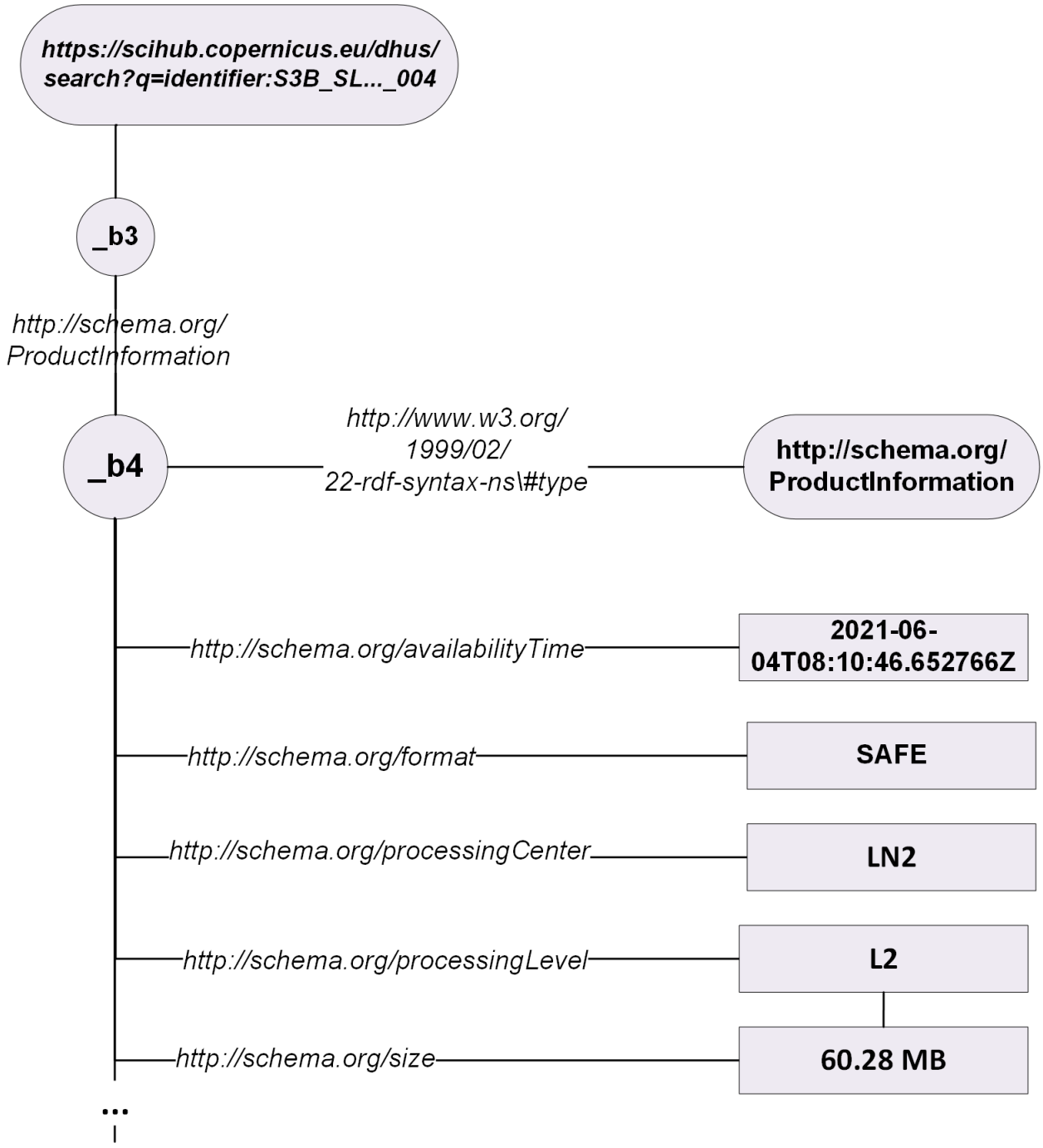

Figure 8: ProductInformation node

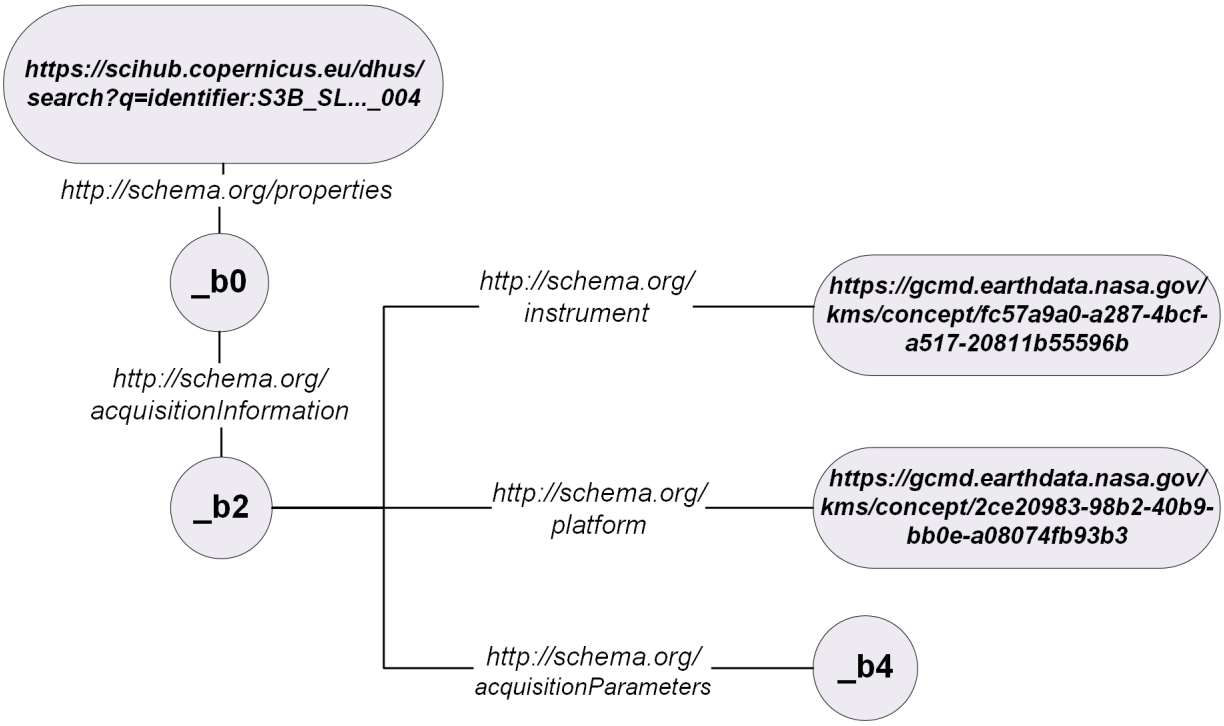

Figure 9: AcquisitionInformation node

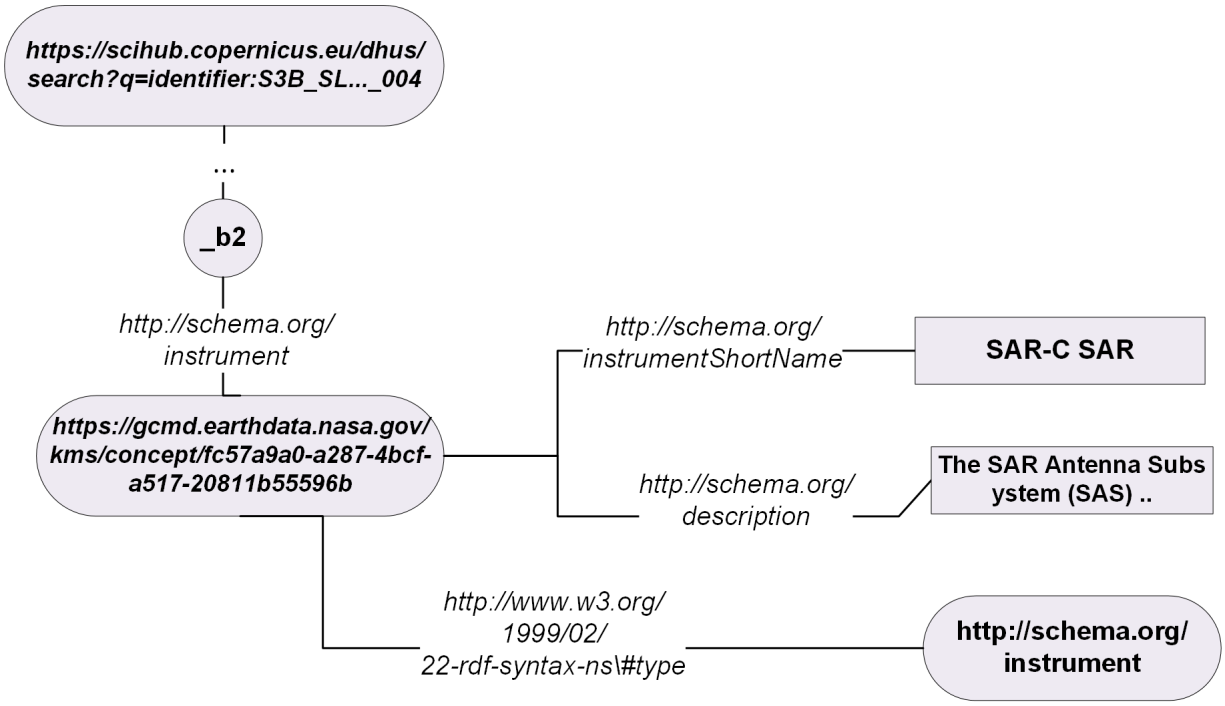

Figure 10: Instrument node

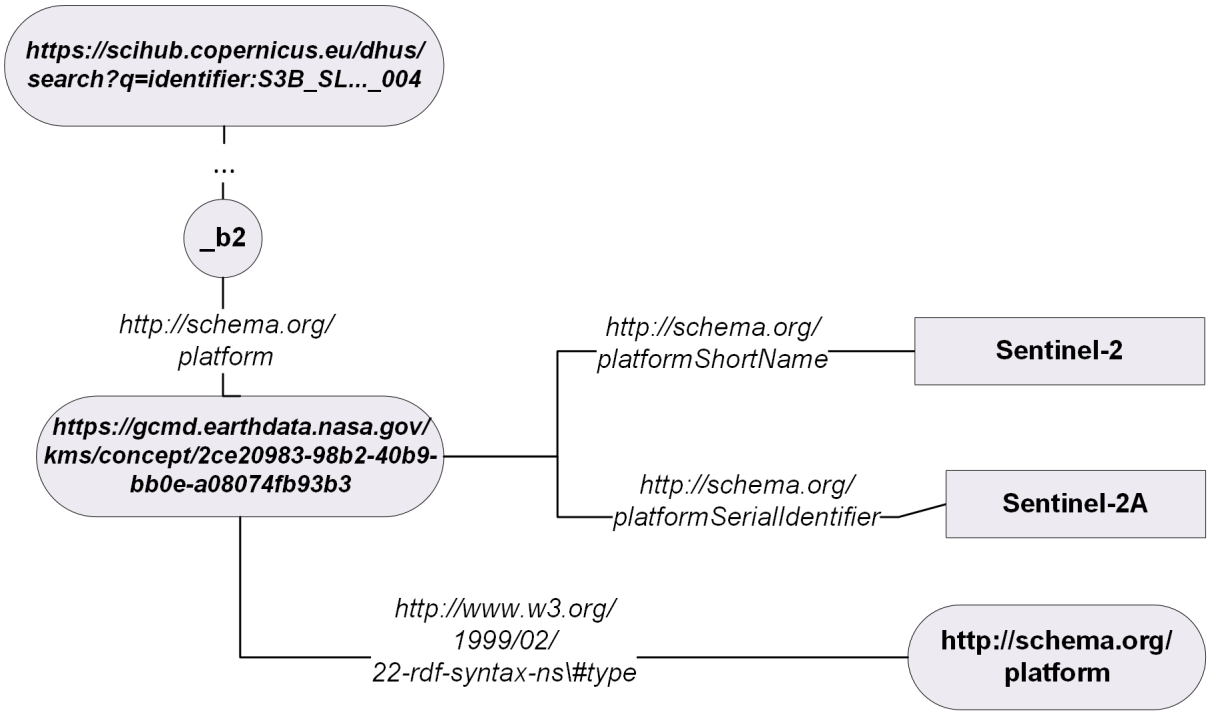

Figure 11: Platform node

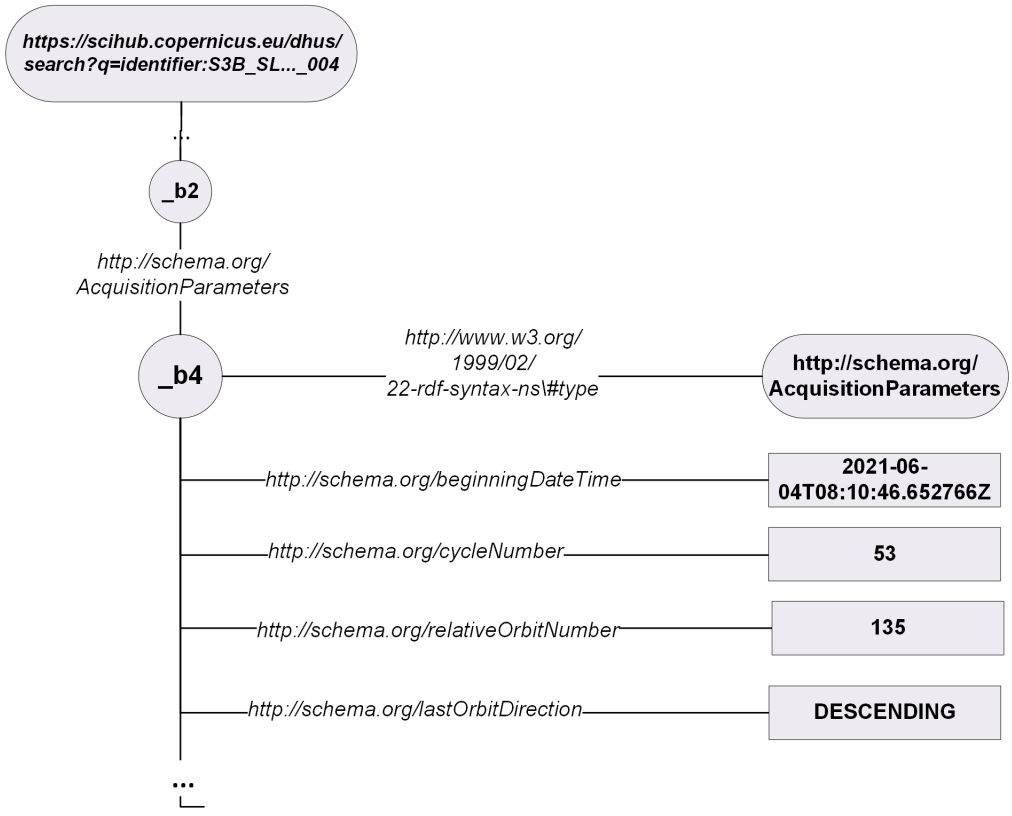

Figure 12: AcquisitionParameters node

3.4.3 The *geometry* node

The Geometry node consists of two optional properties, ”type” and ”coordinates”. Type property can be connected to the geometry node with a simple triplet. Coordinates though introduce the necessity of RDF collections[8]. Coordinates are neither a simple URIREF, nor a blank node or a Literal.

An RDF collection is a group of things represented as a list structure in the RDF graph. This list structure is constructed using a predefined collection vocabulary consisting of the predefined type rdf:List, the predefined properties rdf:first and rdf:rest, and the predefined resource rdf:nil. Each member of the collection, such as 31.499 in [31.499, 21.3229] is the object of an rdf:first property whose subject is a blank node, that represents a list. This list resource is linked to the rest of the list by an rdf:rest property. The end of the list is indicated by the rdf:rest property having as its object the resource rdf:nil (the resource rdf:nil represents the empty list, and is defined as being of type rdf:List).

The complexity of the coordinates' list depends on the type of geometry (e.g. POLYGON, MULTIPOLYGON etc.). All Copernicus products where identified to have only ”POLYGON” as a possible value. According to OGC 17-003r2 a Polygon is represented by a list of items with each item being a list of exactly two numeric values, or a point. E.g.:

```
"geometry": {
      "coordinates": [
        [
           [-2.682513, 63.261372],
           [-2.695740, 61.997604],
           [0.005087, 61.965195],
           [0.135472, 63.227173],
           [-2.682513, 63.261372]
        ]
      ],
      "type": "Polygon"
}
```

3.5 Summary

In this chapter we present the process of annotating the metadata of an EO dataset, following the OGC 17-003r2 standard. We describe the construction of an RDF graph which will lead us to the desired JSON-LD encoding.

[8] https://www.w3.org/TR/rdf-primer/#collections

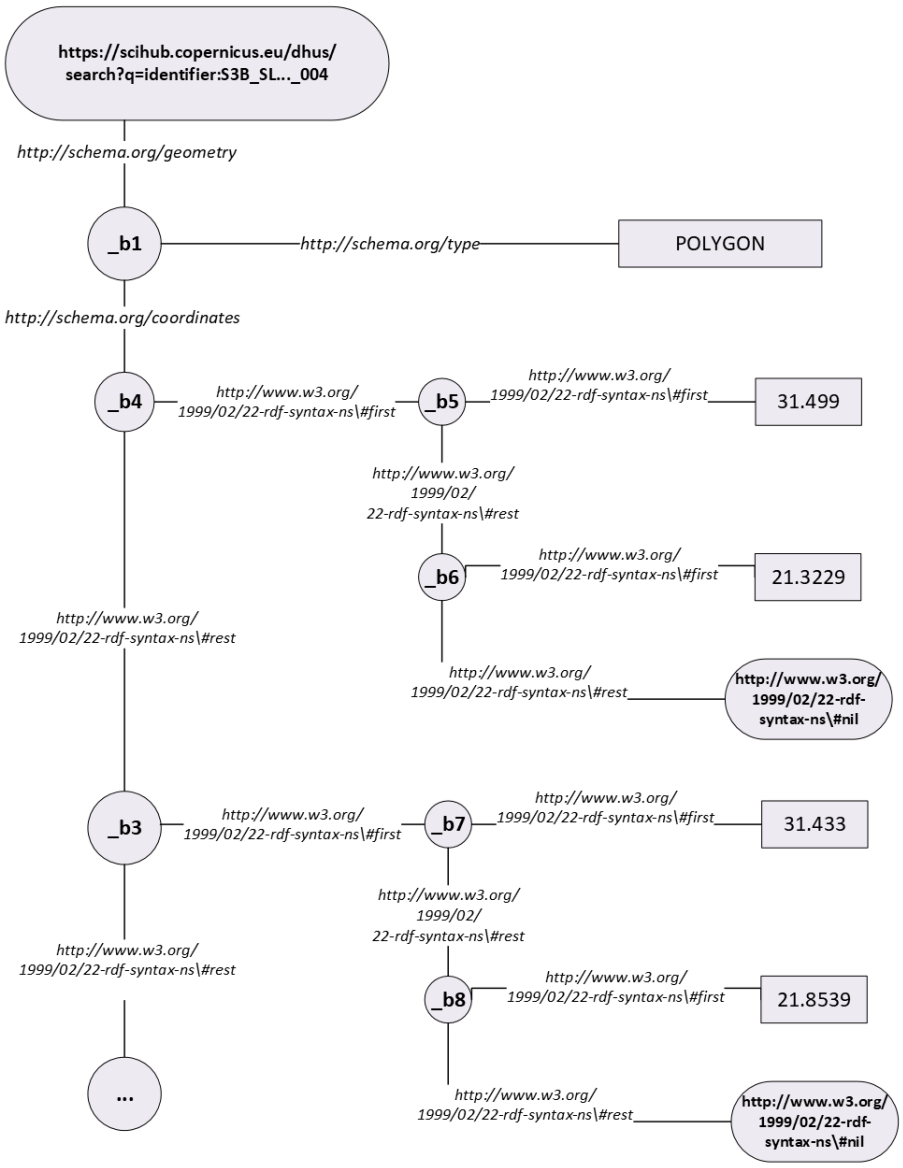

Figure 13: Geometry node

4. IMPLEMENTATION

In this chapter we describe the implementation procedure that we followed. We delve in the development steps and techniques needed to do for the production of our API and User Interface. Also, we briefly present the technologies we chose to use for the implementation.

4.1 API

This section has the purpose of presenting the final API for automating the EO dataset annotation procedure. Firstly, we describe the development language we chose to use, as well as the libraries it provides us. After the technical description, we move on to the APIs' functionality where we cite examples of the code implementation and the algorithms we developed for the automation. We also present example requests that our server accepts and can be consumed by others.

4.1.1 Technologies

4.1.1.1 Python

Python is a popular general-purposed programming language. It stands out for its readability and resemblance to natural language. This has made it one of the most commonly used languages, while numerous libraries are written in it. Python is also known for dealing with JSON objects and annotations as it comes with a built-in package, called *json*, for encoding and decoding JSON data.

We chose to implement our automation procedure in Python because of the packages it provides. *PyLD*[1], *json*[2], *Requests*[3] are the main packages we used to produce the final JSON-LD document, as described in Chapter 3, which contains the annotated EO data. These libraries will be described in the following subsections.

4.1.1.2 Flask

Flask is a Web Server Gateway Interface (WSGI)[4] microframework written in Python. It is described as a microframework, because it does not require any dependencies. It is designed as a web framework for RESTful API development. Flask is higly configurable and combines easily with extensions which offers the developers a large scale of options for how users access data. Since we do not need to store any information concerning the user's request or add any level of authentication, Flask is an ideal option for our server. Flask is available for Python 3 and can be easily installed with Python's official package manager pip.

[1] `https://github.com/digitalbazaar/pyld`
[2] `https://www.w3schools.com/python/python_json.asp`
[3] `https://pypi.org/project/requests/`
[4] `https://wsgi.readthedocs.io/en/latest/`

4.1.2 Libraries

4.1.2.1 PyLD

PyLD is a JSON-LD processor written in Python. As mentioned in its documentation, this library aims to conform with the following:

1. JSON-LD 1.1[5],
2. JSON-LD 1.1 Processing algorithms and API[6] and
3. JSON-LD 1.1 Framing[7]

With that in mind, we can use PyLD's implementation to produce our JSON-LD document from an RDF graph as it fully conforms to the offical JSON-LD specifications.

4.1.2.2 Python Requests

To communicate with Copernicus Open Access API we use Python's package *Requests* which enables us to send the needed HTTP requests easily.

4.1.2.3 JSON encoder

Python has a built-in package, called *json*, which enables the encoding and decoding of JSON data as defined in RFC-7159.[8]

4.1.3 Functionality

Our Flask API serves as a proxy between the user and the API of Copernicus Open Access Hub. Authentication of the user's credentials is handled entirely by the mentioned API, while our server is only responsible for processing and annotating the metadata retrieved from it.

Our server's life cycle begins with an HTTP request at the */api/products/<Product-Guid>* endpoint of our server. This endpoint accepts GET requests with optional Basic Authentication. Sentinel-5P products can be fetched with the website's default credentials, thus authentication is optional.

An optional query parameter is supported to specify whether the requested product id is a product of Sentinel-5P or not. If this parameter is omitted, it is considered as false and the server will proceed to request a Sentinel-1, Sentinel-2, or Sentinel-3 product. This is required because a different request is made for Sentinel-5P products.

[5] https://www.w3.org/TR/json-ld11/
[6] https://www.w3.org/TR/json-ld11-api/
[7] https://www.w3.org/TR/json-ld11-framing/
[8] https://datatracker.ietf.org/doc/html/rfc7159.html#section-1

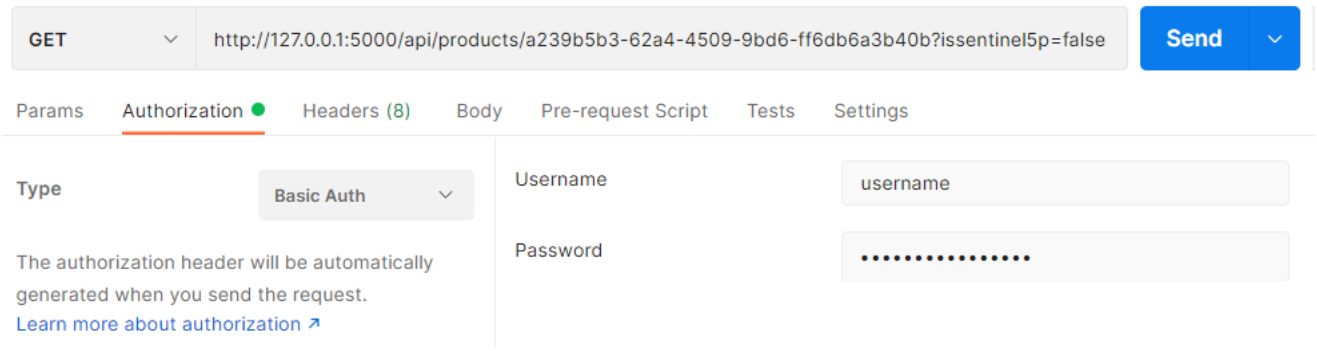

Figure 14: Example request of a Sentinel-1 product

```python
from flask import Flask, jsonify, request
from flask_cors import CORS
from services.requestExecuter import RequestExecuter

app = Flask(__name__)
CORS(app)

@app.route('/api/products/<product_id>', methods=['GET'])
def getProduct(product_id):
  isSentinel5P = False
  if(request.args.get('issentinel5p')):
    isSentinel5P = request.args.get('issentinel5p').lower() == 'true'

  username = request.authorization.username
  password = request.authorization.password

  result = RequestExecuter().executeRequest(product_id, isSentinel5P, username, password)

  response = {
    "result": result
  }

  return jsonify(response)
```

Figure 15: API endpoint implementation

Once the request is received in the server side, a request is made to the Copernicus API Hub as described in Chapter 3.1

```python
class OpenAccessHubService:
    apihubUrl = 'https://apihub.copernicus.eu/apihub/odata/v1/Products(\'{}\')/Attributes?$format=json'
    s5phubPUrl = 'https://s5phub.copernicus.eu/dhus/odata/v1/Products(\'{}\')/Attributes?$format=json'

    def getProductData(self, username, password, productId, isSentinel5P):
        self.session = requests.Session()
        self.url = ""

        if(isSentinel5P):
            self.session.auth = ("s5pguest", "s5pguest")
            self.url = OpenAccessHubService.s5phubPUrl.format(productId)
        else:
            self.session.auth = (username, password)
            self.url = OpenAccessHubService.apihubUrl.format(productId)

        resp = self.session.get(
            self.url,
            auth=self.session.auth
        )
        resp.raise_for_status()
        return resp.json()
```

Figure 16: Request to Copernicus Open Access Hub API

Copernicus Open Access API responds with a JSON encoding, as requested. The response contains the metadata of the Sentinel product that are to be annotated by our tool. After fetching the metadata, we collect all the available attributes and annotate them according to the annotations explained in Chapter 3.3. To accomplish that, we use a configuration file (Figure 17) that contains the mappings between the Copernicus and OGC 17-003r2 properties. The same configuration file will be used later in the request's lifecycle to construct the needed graph. The goal of the configuration file is to allow future vocabulary expansions or changes in the annotated values.

Although the annotation of the properties varies between different Sentinel product types, the same configuration file is used no matter the product type requested. As it is shown in Figure 17 if a property is annotated by more than one Copernicus properties, all the possible matches of the property are placed into an array. The first that will be found among the attributes retrieved from Copernicus will be stored.

The values of annotated properties are then formatted according to OGC 17-003r2's accepted values.

Example 1:

For example we get the following:

Copernicus value: *<gml:coordinates>36.1275,17.2414 34.1716,16.8516 33.9029, 19.4971 </gml:coordinates>*

This Copernicus value is converted to the OGC 17-003r2's accepted value as shown below:

Accepted value: *[[-2.682513, 63.261372], [-2.695740, 61.997604], [0.005087, 61.965195]]*

Example 2:

An other example is the following:

Copernicus value: *Land OLCI Processing and Archiving Centre [LN1]*

Accepted value: *LN1*

The last values that we need to collect are the IDs of the *Instrument*, *Platform* and *EarthObservation* nodes. For the *Instrument* and *Platform* IDs, we use the following URI: *https://gcmd.earthdata.nasa.gov/kms/concept/<Concept_Id>*, where *Concept_Id* is the ID of each instrument or platform. The IDs of all the available instruments and platforms can be found on:

`https://gcmd.earthdata.nasa.gov/kms/concepts/concept_scheme/instruments/?format=json` and `https://gcmd.earthdata.nasa.gov/kms/concepts/concept_scheme/platforms/?format=json` respectively.

```
{
    "eoEarthObservation": {
        "id": "",
        "geometry": {
            "type": "Footprint",
            "coordinates": "Footprint"
        },
        "eoProperties": {
            "status": "Status",
            "title": "Identifier",
            "identifier": "Identifier",
            "date": "Date",
            "created": "Ingestion Date",
            "available": ["Generation Date", "Creation Date"],
            "eoAcquisitionInformation": {
                "eoAcquisitionParameters": {
                    "acquisitionType": "Acquisition Type",
                    "acquisitionSubType": "Mission datatake id",
                    "relativeOrbitNumber": "Relative orbit (start)",
                    "tileId": "Tile Identifier",
                    "cycleNumber": "Cycle number",
                    "operationalMode": ["Instrument mode", "Mode"],
                    "swathIdentifier": "Instrument swath",
                    "polarisationChannels": "Polarisation",
                    "orbitDirection": ["Orbit Direction (start)", "Pass direction"],
                    "lastOrbitDirection": ["Orbit Direction (stop)", "Orbit Direction (start)", "Pass direction"],
                    "orbitDuration": "PDU Duration (s)",
                    "orbitNumber": "Orbit number (start)",
                    "lastOrbitNumber": ["Orbit number (stop)", "Orbit number (start)"],
                    "beginningDateTime": "Sensing start",
                    "endingDateTime": "Sensing stop",
                    "eoAcquisitionAngles": {
                        "illuminationAzimuthAngle": "Illumination Azimuth Angle",
                        "illuminationZenithAngle": "Illumination Zenith Angle"
                    }
                },
                "eoPlatform": {
                    "id": "",
                    "platformShortName": ["Satellite name", "Satellite"],
                    "platformSerialIdentifier": "Platform serial identifier",
                },
                "eoInstrument": {
                    "id": "",
                    "instrumentShortName": ["Instrument abbreviation", "Instrument short name"],
                    "description": ["Instrument description text"]
                }
            },
            "eoProductInformation": {
                "productType": "Product type",
                "size": "Size",
                "availabilityTime": "Ingestion Date",
                "timeliness": "Timeliness Category",
                "productGroupId": "Product class",
                "format": "Format",
                "processingMethodVersion": ["Baseline Collection", "Processing baseline"],
                "processingCenter": "Processing Facility Name",
                "processingLevel": ["Processing Level", "Product level"],
                "processingMode": "Processor mode abbreviation",
                "processorVersion": "Processor version",
                "cloudCover": ["Cloud cover percentage", "Cloud Cover Percentage (%)"],
                "snowCover": ["Snow ice percentage", "Snow or Ice Cover Percentage (%)"]
            }
        }
    }
}
```

Figure 17: Graph Definition - Annotation configuration

The next step is to construct an RDF Graph as described in Chapter 3.4. This graph conforms with the standards defined by the JSON-LD specification[9]. The main classes we constructed to represent the Graph are the following: *Node, Literal, Graph, EOGraph*. Node objects are used to represent URIREF or Blank nodes. As we can see in Listing 1, when a node is created without a value, an auto-generated value is used instead. This value will also be used as an identifier for blank nodes. *Literal* objects have one additional property representing their datatype.

[9] https://www.w3.org/TR/json-ld-api/

```
class Node:
    count = 0
    def __init__(
        self,
        nodeType,
        nodeValue=None,
    ):
    self.type = nodeType
    self.value = nodeValue

    if(self.value==None):
        self.value = "_:b{}".format(Node.count)
        Node.count+=1

def serialize(self):
    return {
        "type": self.type,
        "value": self.value
    }
```

Listing 1: Class Node

```
class Literal(Node):
 self.datatypeUri = "http://www.w3.org/2001/XMLSchema#{}"
 def __init__(
    self,
    value,
 ):
    xmlType = mapToXmlType(value)
    self.type = "Literal"
    self.value = value
    self.datatype = self.datatypeUri.format(xmlType)

 def serialize(self):
    return {
        "type": self.type,
        "value": self.value,
        "datatype": self.datatype
    }
```

Listing 2: Class Literal

Triplets of *Node* and *Literal* objects construct a Graph.

```
class Graph:
    def __init__(self):
        self.triples = []

    def addTriple(self, subject, predicate, object):
        triple = {}
        triple["subject"] = subject.serialize()
        triple["predicate"] = predicate.serialize()
        triple["object"] = object.serialize()

        self.triples.append(triple)
```

Listing 3: Class Literal

The class *EOGraph* inherits from the class *Graph*. Though *EOGraph* class has the additional function of constructing the RDF graph for the annotated data of Copernicus products. Given a dictionary with all the annotated properties and their values, and a graph definition we construct recursively the requested structure. In Algorithm 1 we can see the algorithm we produced to automate the process of the RDF construction.

Algorithm 1 Add Earth Observation Triples

1: **procedure** AddEoTriples($graph, values, definition, subject = None$)
2: **for** key in $definition$ **do**
3: $predicate \leftarrow UriRef(values[key])$
4: $currentNode \leftarrow definition[key]$
5: **if** $typeof(currentNode) = string$ **then**
6: $object \leftarrow Literal(values[key])$
7: $graph.AddTriple(subject, predicate, object)$
8: **else if** $typeof(currentNode) = array$ **then**
9: $graph.AddCollection(values[key], currentNode)$
10: **else if** $typeof(currentNode) = dictionary$ **then**
11: **if** $currentNode.hasProperty("id")$ **then**
12: $object \leftarrow UriRef(values[key])$
13: **else**
14: $object \leftarrow BlankNode()$
15: **end if**
16: $graph.AddTriple(object, Node(type), Node(key))$
17: **if** $subject \neq None$ **then**
18: $graph.AddTriple(subject, predicate, object)$
19: **end if**
20: $graph.AddEoTriples(values, currentNode, object)$
21: **end if**
22: **end for**
23: **end procedure**

Once our *EOGraph* object is filled with its triples, it is parsed into a PyLD graph object. After this point, we can take advantage of all the features supported by JSON-LD, such as compacting, flattening, framing, etc.

As we mentioned in section 2.2, JSON-LD serves as a more human readable way to represent data and data relations. We tried to make the final represantation of the JSON-LD document as simple as possible. For this reason, we used JSON-LD Framing [10] to force a specific tree layout to our JSON-LD document.

Framing, besides shaping data, also introduces the use of a context to compact data. As descibed in JSON-LD specification[11], *"When two people communicate with one another, the conversation takes place in a shared environment, typically called "the context of the conversation". This shared context allows the individuals to use shortcut terms, like the first name of a mutual friend, to communicate more quickly but without losing accuracy. A context in JSON-LD works in the same way. It allows two applications to use shortcut terms to communicate with one another more efficiently, but without losing accuracy."*. In our implementation we use the context published by Schema.org[12]. In Listing 4 we can see the frame used for the JSON-LD production.

```
{
  "@context": {
    "@vocab": "http://schema.org/",
    "coordinates": {
      "@container": "@list"
    }
  },
  "properties": {
    "acquisitionInformation": {
      "instrument": {},
      "platform": {},
      "acquisitionParameters": {
        "acquisitionAngles": {}
      }
    },
    "productInformation": {}
  }
}
```

Listing 4: Frame of JSON-LD document

The JSON-LD document is then returned as the response of the user's request.

[10] https://www.w3.org/TR/json-ld11-framing/

[11] https://json-ld.org/spec/latest/json-ld/#the-context

[12] https://schema.org/docs/jsonldcontext.json

4.2 Web Client

This section has the purpose of presenting the Web Client we implemented. Firstly we describe the web development tools we chose to use. After the technical description, we move on to the web Client's functionality where we give examples of the interface and how a user can interact with it and produce the desired JSON-LD document of a sentinel product.

4.2.1 Technologies

4.2.1.1 React Typescript

React[13] is a declarative open-source JavaScript library created by Facebook, that is used for Web Development purposes such as user interfaces (UIs) for applications. It uses small pieces of code called components which help developers build high-quality UIs for their web applications.

TypeScript[14] is JavaScript with syntax for types. It allows us to write more readable and testable code and build a safety net with type checking.

We chose to implement our API's web client with React and Typescript as their combination lets us easily create a high-quality UI and provides a great number of tutorials and support from its community. The development tools that React and Typescript provide, also help speed up the development process.

4.2.1.2 Material UI

Material UI[15] is a library that provides and imports React components. This helps developers speed up their work since React contains a set of advanced components for creating applications in the most efficient way as described in section 4.2.1.1.

4.2.2 Interface

The user who is interested in getting a GeoJSON-LD document in a more easy and straight-forward way can do so by using our API's UI. When accessing the Web Client the user is prompted to fill in some information, which will give him/her the data he/she needs. An example of the start page is presented in Figure 18.

[13] `https://reactjs.org`
[14] `https://www.typescriptlang.org/`
[15] `https://mui.com/`

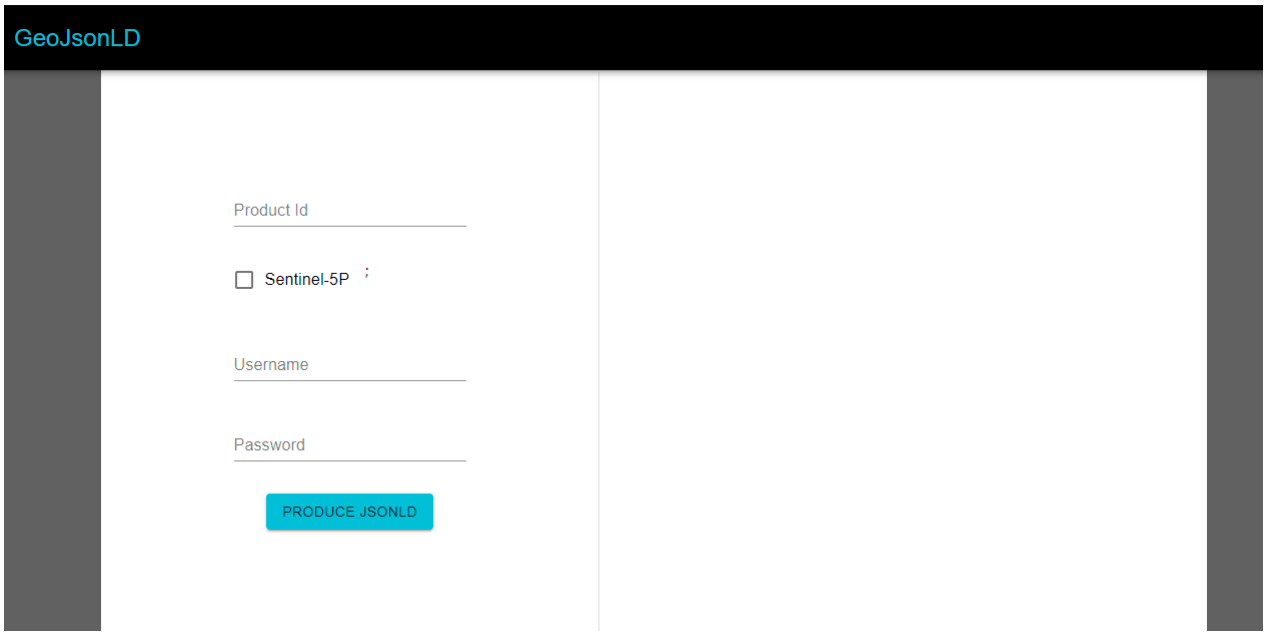

Figure 18: Web Client Start Page

As we can see the first information that the UI requests is the *Product Id*. This is the Id that we get from Copernicus Hub and describes uniquely the image for which GeoJSON-LD document will be produced. *Product Id* must be a Universally Unique Identifier(UUID). A UUID is a universally unique identifier (string) that identifies a digital entity that needs to be managed and kept available for a length of time. It is a 128-bit value consisting of one group of 8 hexadecimal digits, followed by three groups of 4 hexadecimal digits each, followed by one group of 12 hexadecimal digits. If the inserted *Product Id* is empty or not a valid UUID an error message describing the error case will appear. We can see an example of each case in Figures 19 and 20.

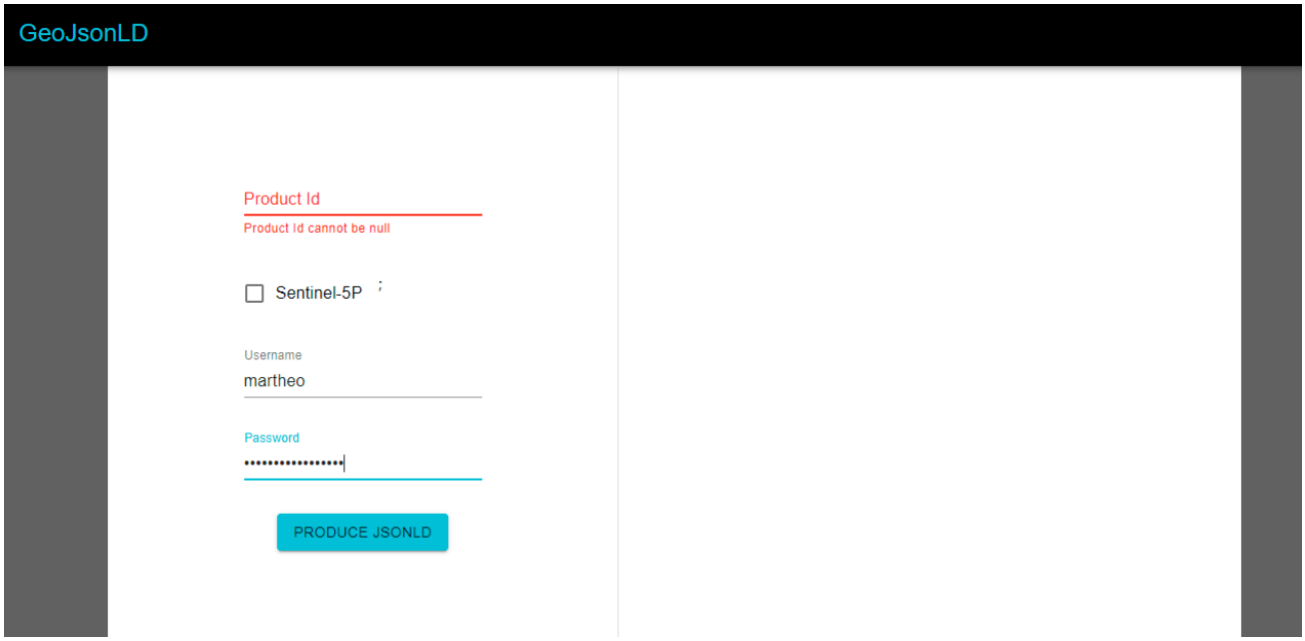

Figure 19: Web Client Error-Empty Id

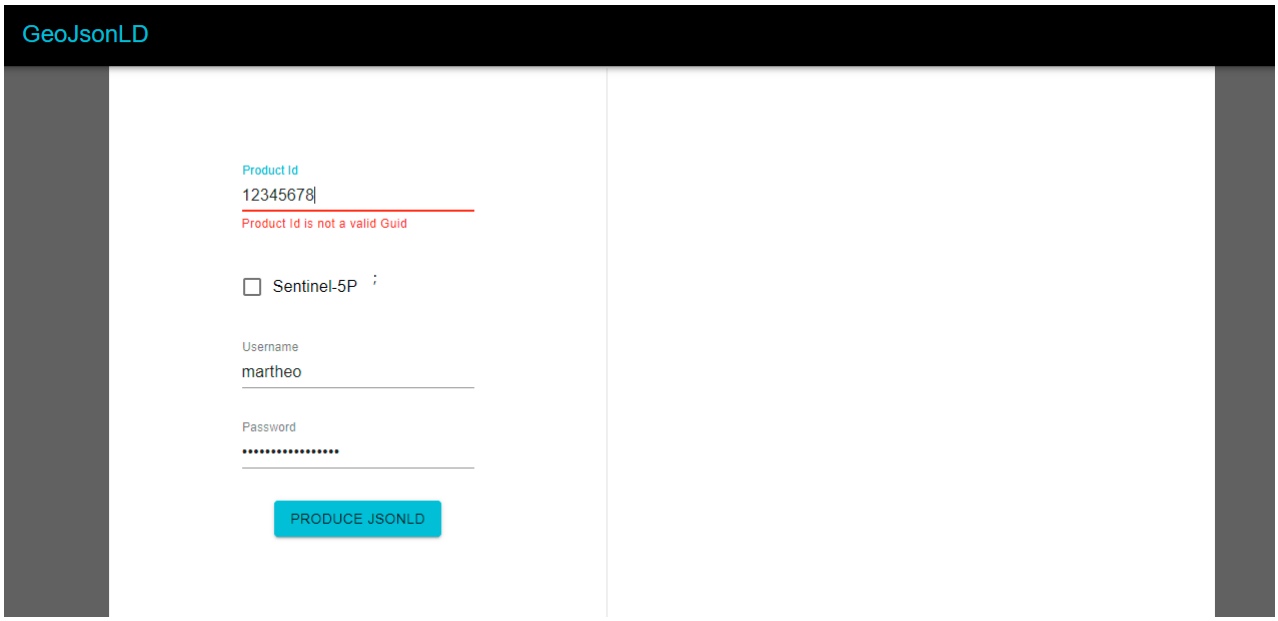

Figure 20: Web Client Error-Invalid Id

The next fields the user needs to fill is *Username* and *Password*. These are the credentials of Copernicus Open Access Hub where the user must maintain an account. When these fields are empty, the equivalent error message is displayed to the user, as shown in Figures 21 and 22.

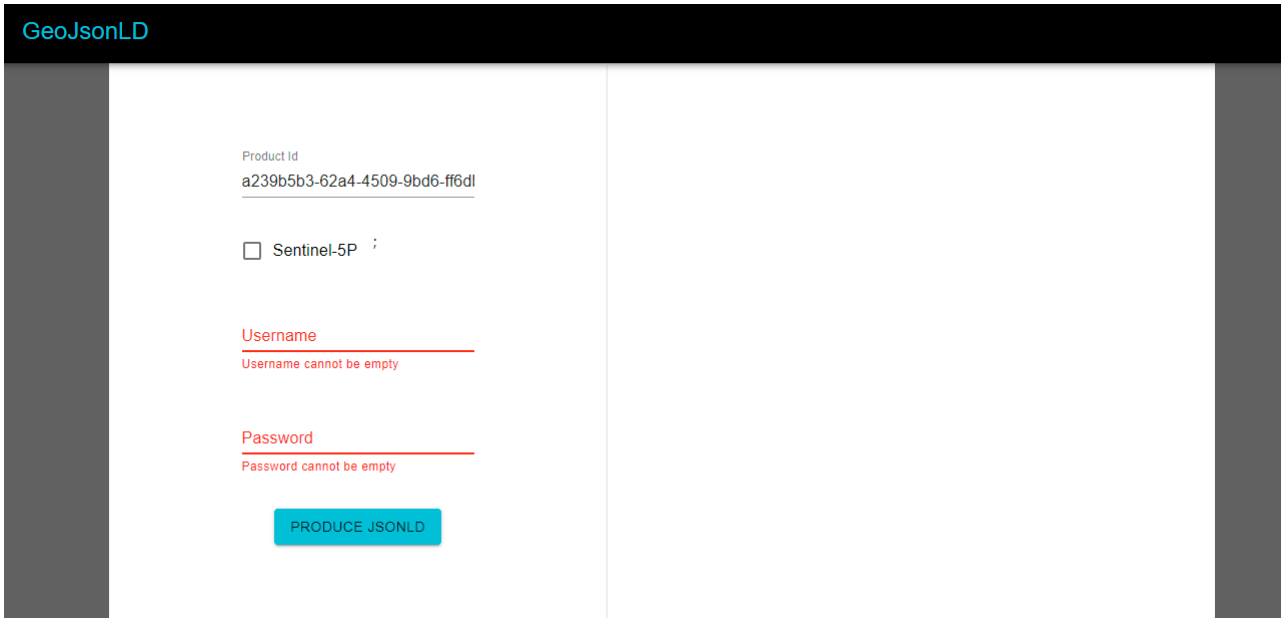

Figure 21: Web Client Error-No Username/Password

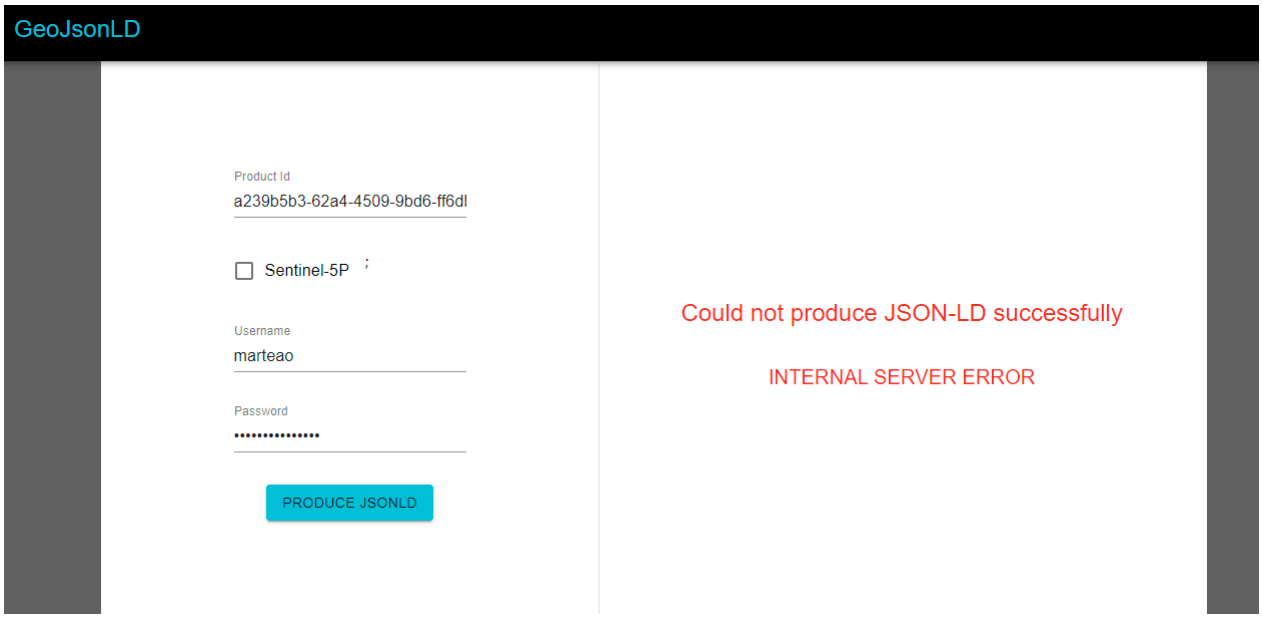

Figure 22: Web Client Error-Incorrect Username/Password or Wrong Sentinel

In the Web Clients' start page the user will also have to check the *Sentinel-5P* box if the image he/she wills to annotate is a product of Sentinel-5P. As we described in section 3.1 the images from Sentinel-1, Sentinel-2 and Sentinel-3 are found under *https://scihub.copernicus.eu/* and the images of Sentinel-5P are found under *https://s5phub.copernicus.eu/*, so this distinction is needed for the API to collect the correct metadata. If the box is incorrectly checked or not checked, the Web Client will also return an error code like shown in Figure 22.

Finally, when all the fields are filled without validation errors, the user can click the *PRODUSE JSONLD* button. After this action the request is sent to our server and the response with the JSON-LD document is served on our interface. In the next Figures (23 and 24) we see the response that the user gets from the Web Client, a JSON-LD document format which he/she can use for EO practices and tasks of his/her willing.

Figure 23: Web Client Response for Sentinel-1, Sentinel-2 and Sentinel-3

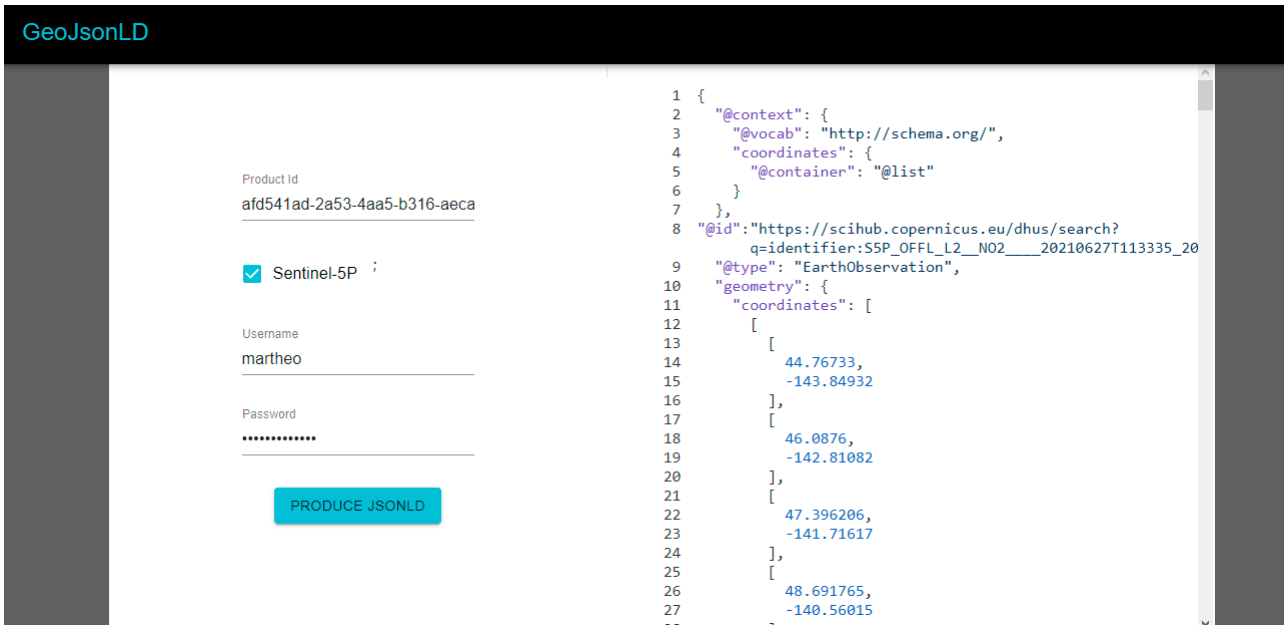

Figure 24: Web Client Response for Sentinel-5P

4.3 Summary

In this chapter we present the implementation steps we followed for the development of the final API and Web Client. The technologies and tools that we used for the implementation are briefly described in each section and we also give examples of how our API and Web Client can be used.

5. CONCLUSIONS AND FUTURE WORK

The purpose of this thesis is to help individuals that use Sentinel products access their GeoJSON-LD annotation.

A next step for this research challenge would be to make the data annotation automation applicable for more Earth Observation Datasets and support all types of geometry structures (e.g. MutliPolygon, MultiPoint etc.). Moreover, our annotation could become more efficient and adaptable in the future with a storage functionality. The requests and annotations of each run could be stored in a database in order to improve the mapping process and track changes, if for example, a new property is added to a Sentinels' product metadata.

At this state the API supports Sentinel-1, Sentinel-2, Sentinel-3 and Sentinel-5P products. The support of all Sentinels that are currently under the Copernicus missions is also a step that could give us a much bigger set of metadata that are valuable for researchers.

Our tool is implemented with the goal to be configurable and extendable at any time. For this reason, a feature that could help users adapt the tool to their needs is to let them provide the definition of the graph and the frame used by the automation.

We have also implemented a UI to facilitate the use of our API. This UI, which is described in section 4.2.2, could be enriched with a geographical search map, to allow users pick their area of interest, and pick between products that correspond to the selected area.

ABBREVIATIONS - ACRONYMS

API	Application Programming Interface
EO	Earth Observation
GUID	Globally Unique Identifier
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
OGC	Open Geospatial Consortium
OWL	Web Ontology Language
O&M	Observations & Measurements
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
UI	User Interface
UMM	Unified Metadata Model
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	Extensible MarkupLanguage

BIBLIOGRAPHY

[1] Fankar Armash Aslam, Hawa Nabeel Mohammed, and Jummal Musab Mohd. Munir Murade Aaraf Gulamgaus. Efficient Way Of Web Development Using Python And Flask, 2015. https://core.ac.uk/download/pdf/55305148.pdf.

[2] Tim Berners-Lee and James Hendler Ora Lassila. The Semantic Web: A New Form of Web Content That is Meaningful to Computers Will Unleash a Revolution of New Possibilities. *Scientific American*, 2001. https://www.researchgate.net/publication/225070375_The_Semantic_Web_A_New_Form_of_Web_Content_That_is_Meaningful_to_Computers_Will_Unleash_a_Revolution_of_New_Possibilities.

[3] Y. Coene and U. Voges O. Barois. OGC EO Dataset Metadata GeoJSON(-LD) Encoding Standard, 2020. https://docs.opengeospatial.org/is/17-003r2/17-003r2.html#45.

[4] Markus Lanthaler and Christian Gütl. On Using JSON-LD to Create Evolvable RESTful Services. *3rd International Workshop on RESTful Design*, 2012. http://www.markus-lanthaler.com/research/on-using-json-ld-to-create-evolvable-restful-services.pdf.

[5] Despina Athanasia I. Pantazi. Encoding and Validation of Earth Observation Metadata using Schema.org and SHACL, 2018.