ESILV
A4-IBO-5

Alban STEFF
Soumaya SABRY

Semantic Web Project

Table des matières

Project requirements

This project is about getting bike rentals station's information through APIs in order to display it to a user. The information you can get are ; station status, id, location (coordinates), capacity and the number of available bikes and bike parks. Since the data is in real time, it's a way for users to know where to park their bike or where to find an available bike in an easy way.

We decided to get data from 3 different French cities: Lyon, Paris and **Rennes**. Since the APIs of Lyon and Paris are also giving the status of the stations of nearby cities, we decided to name Paris as "**Grand Paris**" and Lyon as "**Grand Lyon**", meaning we can get information about sub-cities in these two areas.(cf. "how to use" part)

The main requirement for this project is to understand how JSON, JSON-LD and RDF data formats work, and how to arrange a difference source of JSON data on a unique format. So, the second requirement is to make sure to get information about 3 big French cities so we can unify them to used it. The last requirement is to build an app for the user to find what he needs.

Used technologies

To create our app, we used this stack: Python, Jena Apache on Java, JSON, JSON-LD, RDF, APIs, SPARQL, Jena (riot.bat) and Protégé. You can find more details below:

- **Protégé:** used for creating an ontology to define our data model.
- **Python:** get real-time **JSON** data from **APIs**, then add JSON-LD context to fetched JSON files and finally convert JSON-LD data to **RDF** with JENA "riot.bat".
- **Java:** used Apache Jena to load our ontology model from the .owl file in Java and add the rfd data to this model. Then we created our application for the user, and displayed data with SPARQL queries.

Data structure

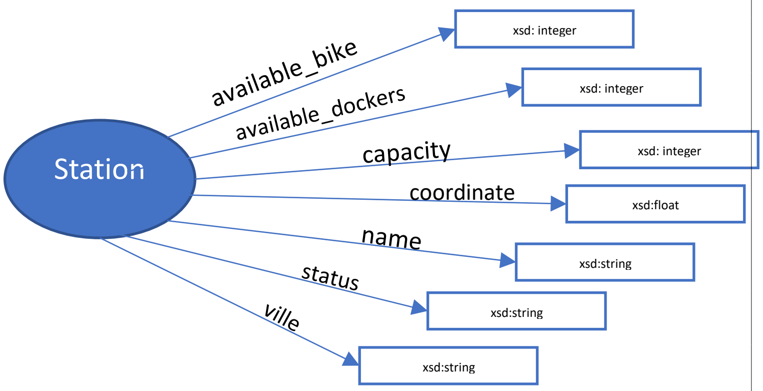
Let's talk about our data structure!

1. RFD & RFDS

First, we decided to look at the JSON data from the APIs, to check and find the different fields we will need for our app. After doing this little work, we selected 7 fields from the JSON files.

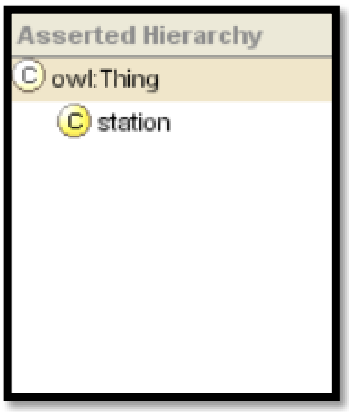
ns:station	rdf:type	rdfs:Class .
ns:available_bike	rdf:type	rdf:Property .
ns: available_bike	rdfs:domain	ns:station .
ns: available_bike	rdfs:range	xsd:integer
ns:name	rdf:type	rdf:Property .
ns:name	rdfs:domain	ns:station .
ns: name	rdfs:range	xsd:string

……

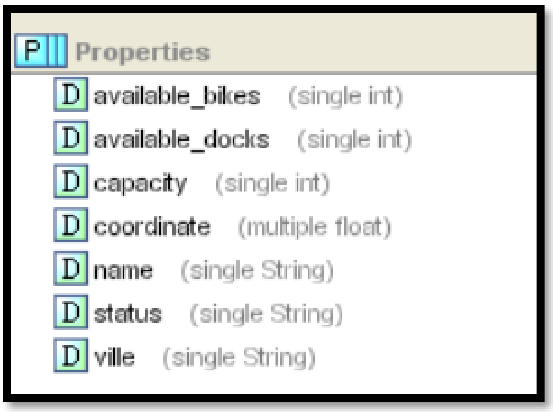


2. Protégé

Then we created an ontology in Protégé based on them:



Asserted Hierarchy
- owl:Thing
  - station



Properties
- available_bikes (single int)
- available_docks (single int)
- capacity (single int)
- coordinate (multiple float)
- name (single String)
- status (single String)
- ville (single String)

We have one class "Station" with 7 DataType properties to structure our data. These properties define our model and represent useful information to show to our app users. We also make sure that each of those properties has Station of a domain.

3. JSON-LD @context

```
"@context": {
        "@vocab": "http://www.owl-ontologies.com/stations-velos.owl#",
        "@base":  "http://www.owl-ontologies.com/stations-velos.owl/"

        "stationcode" :"@id",
        "capacity" :"capacity",
        "numbikesavailable" :"available_bikes",
        "coordonnees_geo" :"coordinate",
        "is_installed" :"status",
        "numdocksavailable" :"available_docks",
        "nom_arrondissement_communes":"ville",

        "datasetid":null, "is_renting":null,"is_returning":null,
        "geometry":null,"ebike":null,"duedate":null,
        "mechanical":null,"record_timestamp":null,
        "recordid":null,"parameters":null,"nhits":null,


        "fields": { "@id": "_:fields", "@container": "@set"} ,
        "records": { "@id": "_:records", "@container": "@list" }

}
```

Here is our JSON-LD context, it is composed it of 4 parts, you can find more details below:

- For the first part, @vocab is used to describe the vocabulary for our ontology and to link the property to our owl using the prefix, and @base is used to link the URI of the stations using the @id corresponding to our ontology.

- For the second part, it used for mapping. On the left is JSON data and on the right is our ontology properties. There, we create a link between JSON files and our ontology.

- For the third part, it is all the data that we don't need, we just write "null" next to it.

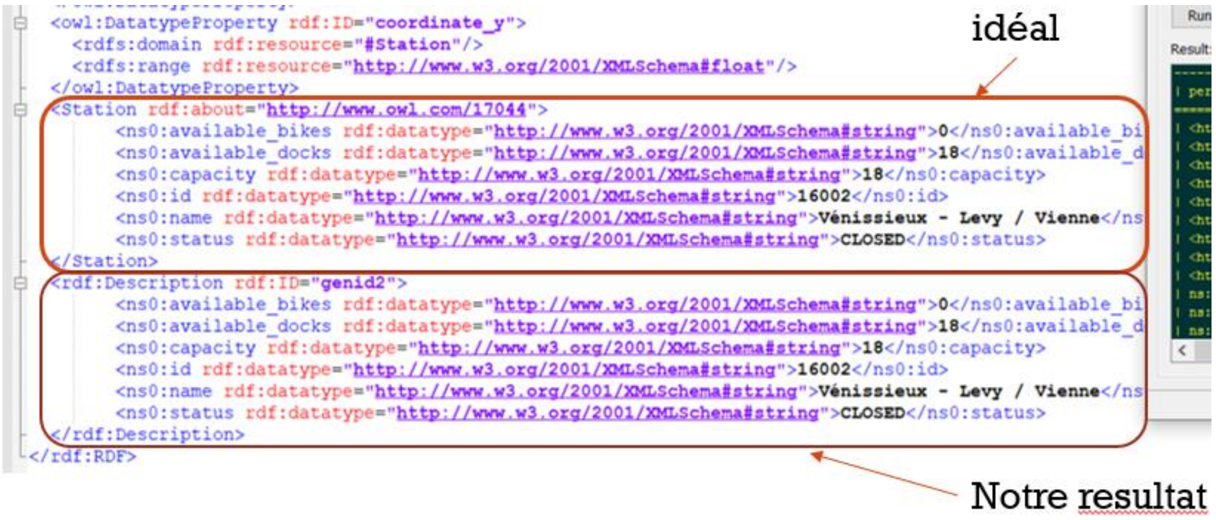
After these parts, the result converted from the JSON-LD to RDF, was not satisfied since we have blank nodes, so we decided to improve our @context with the fourth part.

- For the last part, we define "fields" as a "_: fields" to avoid creating blank nodes with the converter. (same for records)

If converters could understand version 1.1 of JSON-LD, we could also use @nest, but this version is not supported by converters yet.

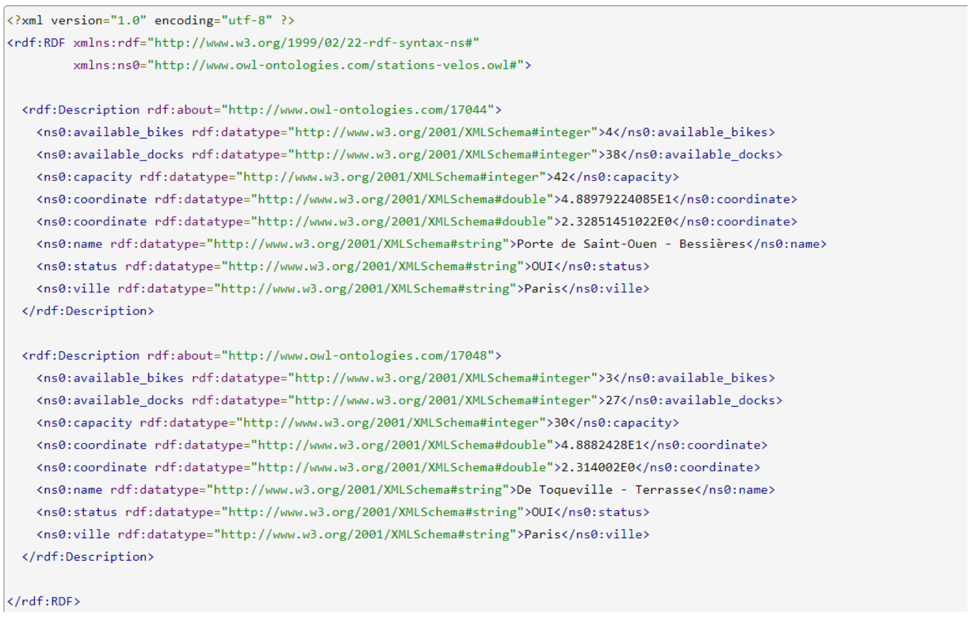
Alban STEFF
Soumaya SABRY

The only problem remaining is each node is starting with a field “Description” instead of “Station” (our ontology class). We can do queries with activating **OWL Inference**. It is working, because the domain of the property is “station”, so it can get that the block is a station.



```
<owl:DatatypeProperty rdf:ID="coordinate_y">
  <rdfs:domain rdf:resource="#Station"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
</owl:DatatypeProperty>
<Station rdf:about="http://www.owl.com/17044">
      <ns0:available_bikes rdf:datatype="http://www.w3.org/2001/XMLSchema#string">0</ns0:available_bi
      <ns0:available_docks rdf:datatype="http://www.w3.org/2001/XMLSchema#string">18</ns0:available_d
      <ns0:capacity rdf:datatype="http://www.w3.org/2001/XMLSchema#string">18</ns0:capacity>
      <ns0:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string">16002</ns0:id>
      <ns0:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Vénissieux - Levy / Vienne</ns
      <ns0:status rdf:datatype="http://www.w3.org/2001/XMLSchema#string">CLOSED</ns0:status>
</Station>
<rdf:Description rdf:ID="genid2">
      <ns0:available_bikes rdf:datatype="http://www.w3.org/2001/XMLSchema#string">0</ns0:available_bi
      <ns0:available_docks rdf:datatype="http://www.w3.org/2001/XMLSchema#string">18</ns0:available_d
      <ns0:capacity rdf:datatype="http://www.w3.org/2001/XMLSchema#string">18</ns0:capacity>
      <ns0:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string">16002</ns0:id>
      <ns0:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Vénissieux - Levy / Vienne</ns
      <ns0:status rdf:datatype="http://www.w3.org/2001/XMLSchema#string">CLOSED</ns0:status>
</rdf:Description>
</rdf:RDF>
```

There is a final result of our data convert from Json-ld to RDF➔



```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ns0="http://www.owl-ontologies.com/stations-velos.owl#">

  <rdf:Description rdf:about="http://www.owl-ontologies.com/17044">
    <ns0:available_bikes rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">4</ns0:available_bikes>
    <ns0:available_docks rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">38</ns0:available_docks>
    <ns0:capacity rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">42</ns0:capacity>
    <ns0:coordinate rdf:datatype="http://www.w3.org/2001/XMLSchema#double">4.88979224085E1</ns0:coordinate>
    <ns0:coordinate rdf:datatype="http://www.w3.org/2001/XMLSchema#double">2.32851451022E0</ns0:coordinate>
    <ns0:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Porte de Saint-Ouen - Bessières</ns0:name>
    <ns0:status rdf:datatype="http://www.w3.org/2001/XMLSchema#string">OUI</ns0:status>
    <ns0:ville rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Paris</ns0:ville>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.owl-ontologies.com/17048">
    <ns0:available_bikes rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">3</ns0:available_bikes>
    <ns0:available_docks rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">27</ns0:available_docks>
    <ns0:capacity rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">30</ns0:capacity>
    <ns0:coordinate rdf:datatype="http://www.w3.org/2001/XMLSchema#double">4.8882428E1</ns0:coordinate>
    <ns0:coordinate rdf:datatype="http://www.w3.org/2001/XMLSchema#double">2.314002E0</ns0:coordinate>
    <ns0:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">De Toqueville - Terrasse</ns0:name>
    <ns0:status rdf:datatype="http://www.w3.org/2001/XMLSchema#string">OUI</ns0:status>
    <ns0:ville rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Paris</ns0:ville>
  </rdf:Description>

</rdf:RDF>
```

4. Real-Temp ♦BONUS♦

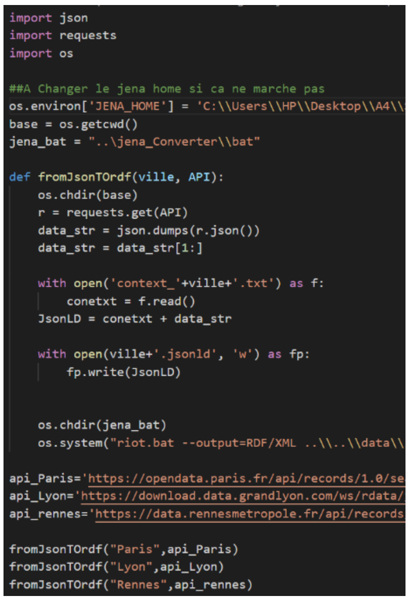
At the end, we created a script to get all the steps automatically:

- Start with setting the environment

/!\the JENA HOME should be change to use our app

- Then getting the JSON data from the APIs
- Adding the context of each city to their JSON
- Saving the result as a JSON-LD file
- Converting the JSON-LD to RDF using the Jena -Riot.bat (same as we use at the TD2)

The script can be easily run on the java project using the *RUNTIME* command, and it is run each time we start the MENU .



```python
import json
import requests
import os

##A Changer le jena home si ca ne marche pas
os.environ['JENA_HOME'] = 'C:\\Users\\HP\\Desktop\\A4\\
base = os.getcwd()
jena_bat = "..\jena_Converter\\bat"

def fromJsonTOrdf(ville, API):
    os.chdir(base)
    r = requests.get(API)
    data_str = json.dumps(r.json())
    data_str = data_str[1:]

    with open('context_'+ville+'.txt') as f:
        conetxt = f.read()
    JsonLD = conetxt + data_str

    with open(ville+'.jsonld', 'w') as fp:
        fp.write(JsonLD)


    os.chdir(jena_bat)
    os.system("riot.bat --output=RDF/XML ..\\..\\data\\

api_Paris='https://opendata.paris.fr/api/records/1.0/se
api_Lyon='https://download.data.grandlyon.com/ws/rdata/
api_rennes='https://data.rennesmetropole.fr/api/records

fromJsonTOrdf("Paris",api_Paris)
fromJsonTOrdf("Lyon",api_Lyon)
fromJsonTOrdf("Rennes",api_rennes)
```

Alban STEFF
Soumaya SABRY

Application architecture

In Zip file, we have 3 folders:

I. **Data**:
   - A Python script "Tempsreel.py" to fetch data from APIs
   - txt files with JSON-LD context for our different cities
   - RDF files with the data

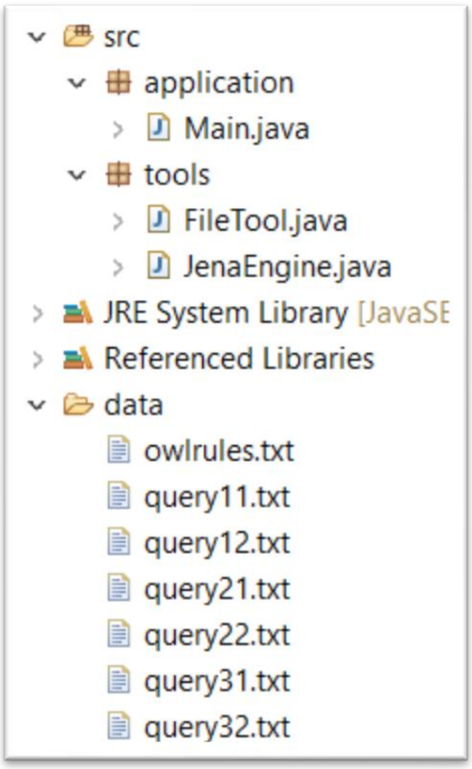
➔We are running the Python script in the Java application.

II. **jena_Converter:** used in the python script to convert the JSON-LD to RDF

III. **Jena_projet :** Our java app, in this, is our app architecture.

We divided it in 2 main folders and 2 library files:

- Source: contains the application and tools to build the app
- Data: contains owlrules to define our model and SPARQL queries
- JRE System Library: Java files
- Referenced Libraries: Jena files



```
src
  application
    Main.java
  tools
    FileTool.java
    JenaEngine.java
JRE System Library [JavaSE
Referenced Libraries
data
  owlrules.txt
  query11.txt
  query12.txt
  query21.txt
  query22.txt
  query31.txt
  query32.txt
```

How to install

To install our app, you need to unzip all files and put them in your Eclipse workspace folder.

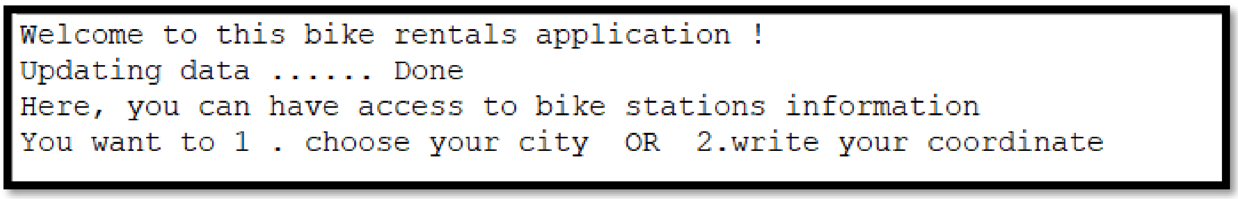
You should change the JENA HOME in Data/Tempsreel.py script.

Then, you can simply launch Main.java in your IDE and follow the instructions on the console!

How to use

When you started the app, you will have some questions displayed on the screen:

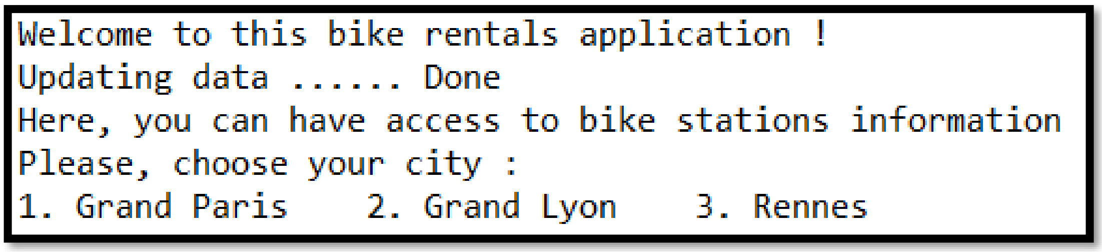
Frist, it asks if you want to choose a city of to write down your coordinate:



```
Welcome to this bike rentals application !
Updating data ...... Done
Here, you can have access to bike stations information
You want to 1 . choose your city  OR  2.write your coordinate
```

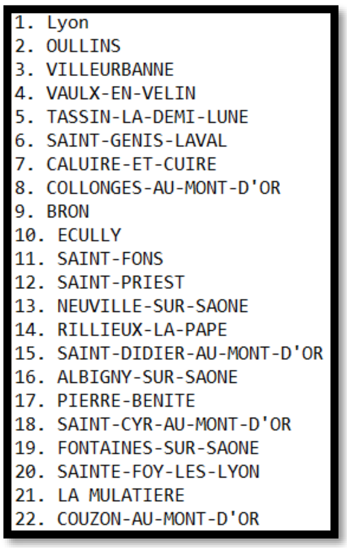
Second, if you choose (1) you get:

- Which city are you interested in? (Grand Paris, Grand Lyon or Rennes)



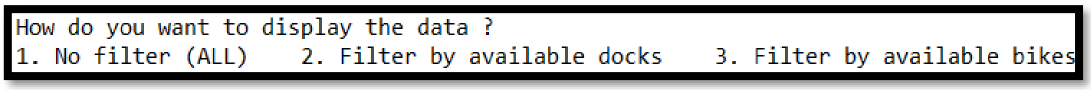
```
Welcome to this bike rentals application !
Updating data ...... Done
Here, you can have access to bike stations information
Please, choose your city :
1. Grand Paris     2. Grand Lyon     3. Rennes
```

- If you choose a "Grand" city, another list of cities is displayed, and you can choose one.



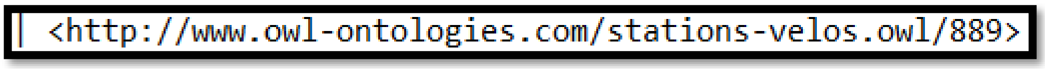
```
1. Lyon
2. OULLINS
3. VILLEURBANNE
4. VAULX-EN-VELIN
5. TASSIN-LA-DEMI-LUNE
6. SAINT-GENIS-LAVAL
7. CALUIRE-ET-CUIRE
8. COLLONGES-AU-MONT-D'OR
9. BRON
10. ECULLY
11. SAINT-FONS
12. SAINT-PRIEST
13. NEUVILLE-SUR-SAONE
14. RILLIEUX-LA-PAPE
15. SAINT-DIDIER-AU-MONT-D'OR
16. ALBIGNY-SUR-SAONE
17. PIERRE-BENITE
18. SAINT-CYR-AU-MONT-D'OR
19. FONTAINES-SUR-SAONE
20. SAINTE-FOY-LES-LYON
21. LA MULATIERE
22. COUZON-AU-MONT-D'OR
```

- After selecting your city, you are asked a way to show the station's data on the screen: unordered, ordered by number of available bikes or by available docks.



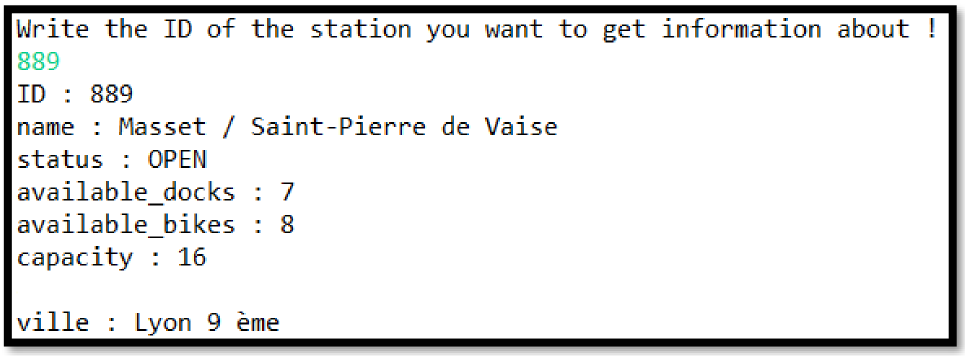
```
How do you want to display the data ?
1. No filter (ALL)    2. Filter by available docks    3. Filter by available bikes
```

- Then, the stations are displayed, and you can write the ID corresponding to the station you want to get information about.



```
| <http://www.owl-ontologies.com/stations-velos.owl/889>
```

➔ The ID is 889!

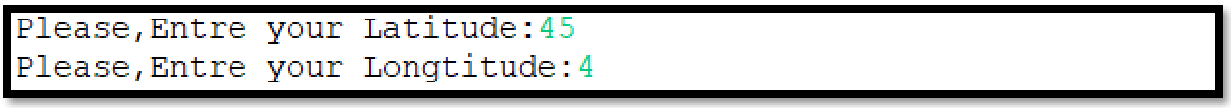


```
Write the ID of the station you want to get information about !
889
ID : 889
name : Masset / Saint-Pierre de Vaise
status : OPEN
available_docks : 7
available_bikes : 8
capacity : 16

ville : Lyon 9 ème
```

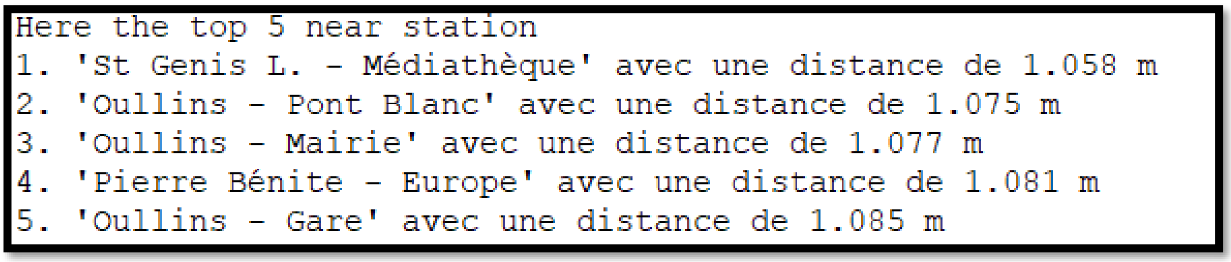
Third, if you choose (2) you get:

- A Question for writing your latitude & longitude:



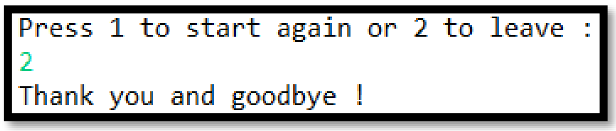
```
Please,Entre your Latitude:45
Please,Entre your Longtitude:4
```

- Then, the top 5 near station are displayed in order:



```
Here the top 5 near station
1. 'St Genis L. - Médiathèque' avec une distance de 1.058 m
2. 'Oullins - Pont Blanc' avec une distance de 1.075 m
3. 'Oullins - Mairie' avec une distance de 1.077 m
4. 'Pierre Bénite - Europe' avec une distance de 1.081 m
5. 'Oullins - Gare' avec une distance de 1.085 m
```

Finally, you are asked if you want to display information about another station or close the app.



```
Press 1 to start again or 2 to leave :
2
Thank you and goodbye !
```