Project Web Datamining&Semantics CheatSheet

A project made by:

- **CAURIER Grégoire**
- **HACHEM Mohamad**
- **BENYEMNA Hamza**

How to Install

- Go to the directory **Jena Fuseki file** and click the fuseki-server.bat to run it.
- Go on http://localhost:3030/
- Go to "manage datasets"
- Click on add new dataset and create a dataset called "Name_you_want" and select the persistent option so you won't have to add the data each time you run **fuseki**.
- Click on **add data** on you're the dataset you selected then on **select files** to select your rdf dataset that are in the **Project datasets** folder and then click on **upload all**.
- Repeat this operation for 4 datasets (call them: EcoleSup, Gares, Velib and Postes)
- Your server **fuseki** is ready don't close it when you execute the python Script.
- Concerning the python project, you need to install : **Flask**, **folium**, **pandas**, and **SPARQLWrapper** with the pip command in the terminal of the project.

How to Use

- Open the **Project** folder in your code editor (We used PyCharm but you can use VSCode if you want)
- Open the **powershell**
- Type in the **powershell python** Project.py and click enter to execute it.
- You can go on the http://127.0.0.1:5000/ but you won't have a nice interface so open the index.html file in the **templates** folder so you will have a nice .css and .html interface.
- Click on the button to see information since we request a lot of data the request can be long to execute.

Report:

Intro:

The aim of this project is to show POI on a map in an easy-to-use UI. To make this, we used gouv.org API because it is a real complete source of data.

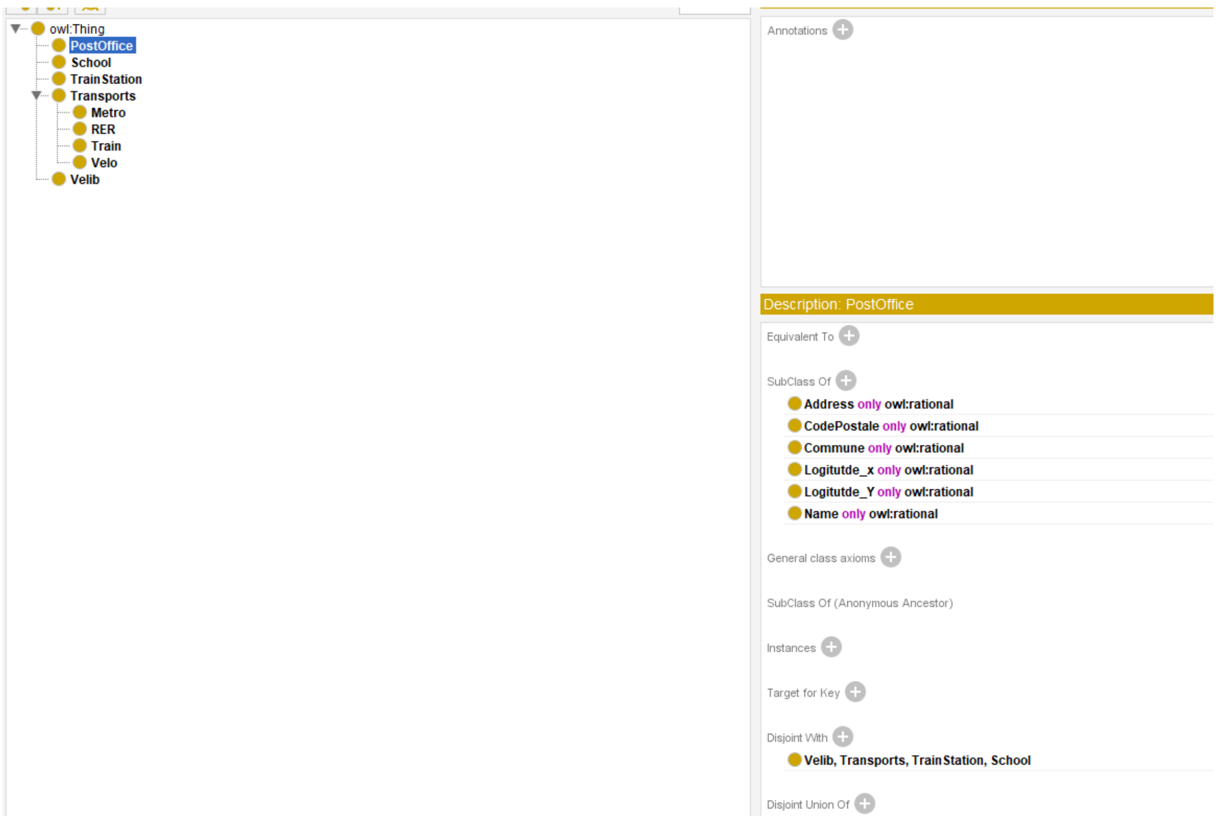
Important notice: we neglect the travellers and trip aspect of the project

Ontology and sources:

We used 4 sources:

- *Schools*
- *Velib*
- *Gares*
- *Postes*

Indeed, we created a simple ontology scheme with one source used for each POI, restrictions and needed properties as you can see in the file **Final_Project_Web_DataSemantics.owl**



```
owl:Thing
    PostOffice
    School
    TrainStation
    Transports
        Metro
        RER
        Train
        Velo
    Velib
```

Annotations

Description: PostOffice

Equivalent To

SubClass Of
- Address only owl:rational
- CodePostale only owl:rational
- Commune only owl:rational
- Logitutde_x only owl:rational
- Logitutde_Y only owl:rational
- Name only owl:rational

General class axioms

SubClass Of (Anonymous Ancestor)

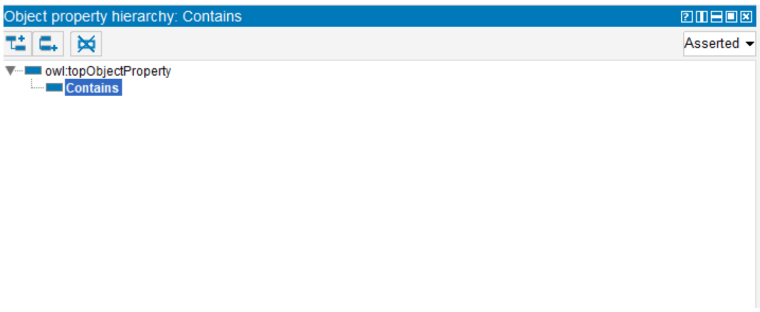
Instances

Target for Key

Disjoint With
- Velib, Transports, TrainStation, School

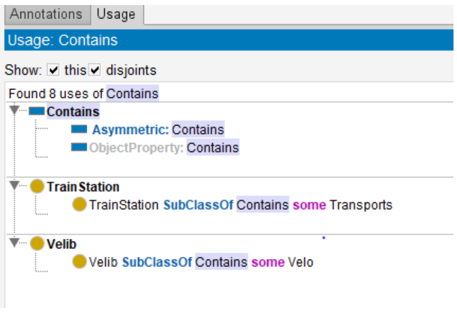
Disjoint Union Of

You can see the **class** hierarchy, organised in **sub classes** with **data** and **object properties** but also disjoin restrictions.


Object property hierarchy: Contains

Asserted

- owl:topObjectProperty
  - Contains

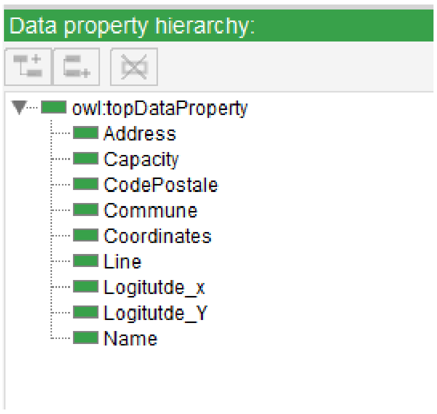

Annotations | Usage

Usage: Contains

Show: ✓ this ✓ disjoints

Found 8 uses of Contains

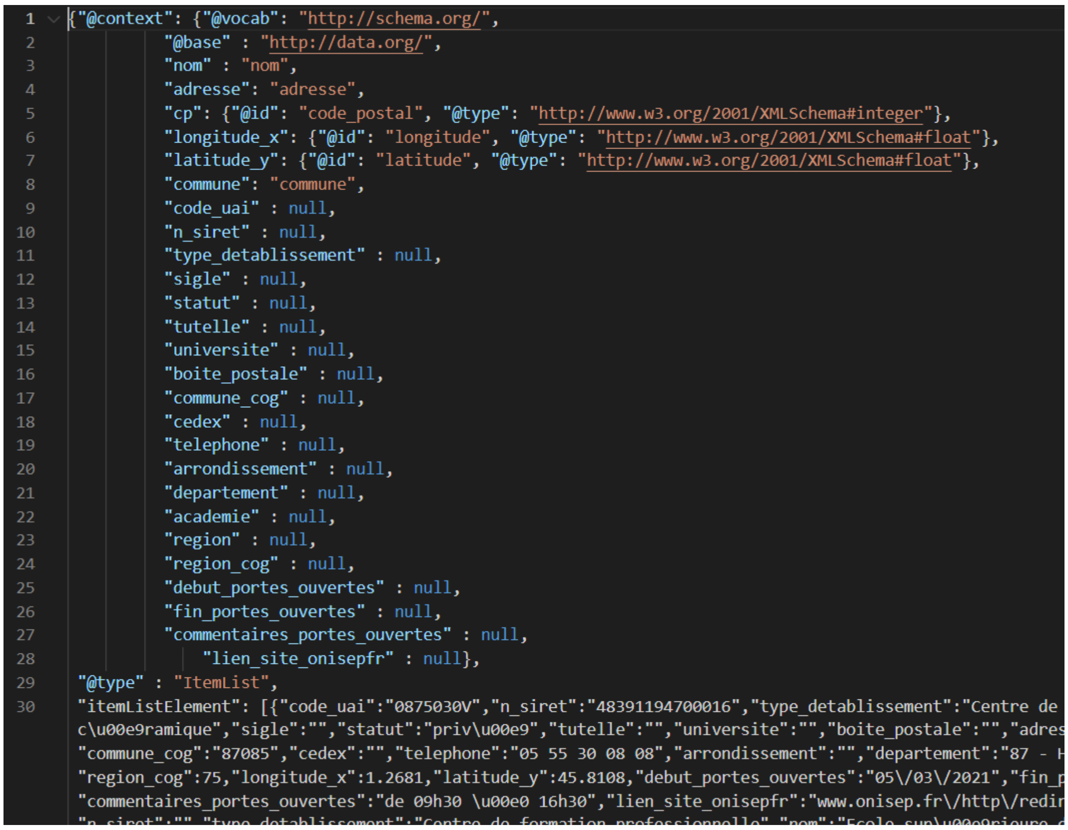
- Contains
  - Asymmetric: Contains
  - ObjectProperty: Contains
- TrainStation
  - TrainStation SubClassOf Contains some Transports
- Velib
  - Velib SubClassOf Contains some Velo


Data property hierarchy:

- owl:topDataProperty
  - Address
  - Capacity
  - CodePostale
  - Commune
  - Coordinates
  - Line
  - Logitutde_x
  - Logitutde_Y
  - Name

Conversion to an rdf file:

We directly used the datasets from data.org that you could find in the Project datasets folder as **jsonld** format. Then, we created a context command line in the files to get only useful properties (mainly name and location). Then, we used **Apache Jena** as in the Lab1 to create our **rdf** files.

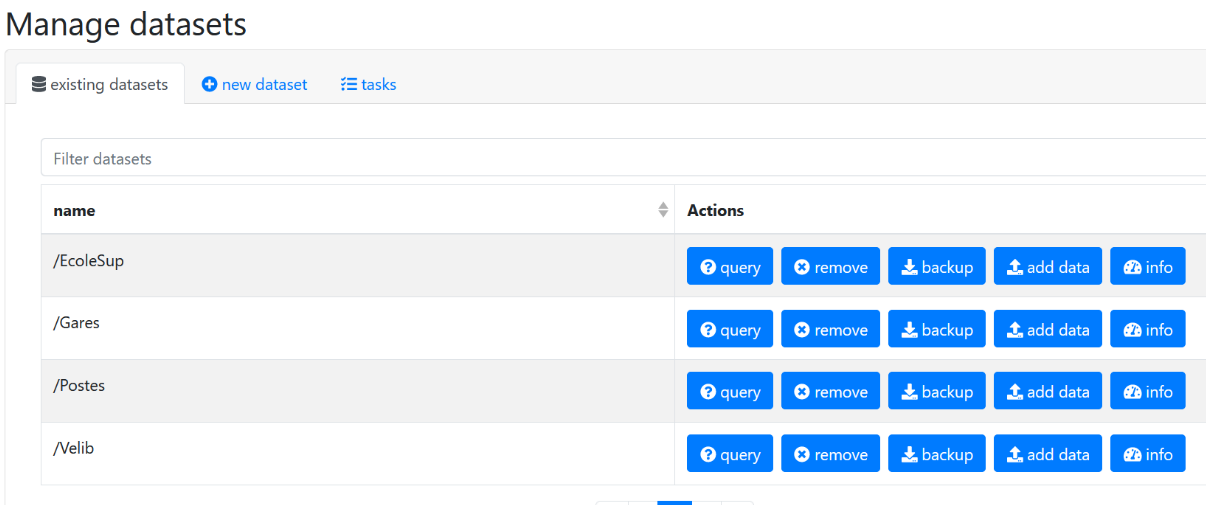

```
{"@context": {"@vocab": "http://schema.org/",
            "@base" : "http://data.org/",
            "nom" : "nom",
            "adresse": "adresse",
            "cp": {"@id": "code_postal", "@type": "http://www.w3.org/2001/XMLSchema#integer"},
            "longitude_x": {"@id": "longitude", "@type": "http://www.w3.org/2001/XMLSchema#float"},
            "latitude_y": {"@id": "latitude", "@type": "http://www.w3.org/2001/XMLSchema#float"},
            "commune": "commune",
            "code_uai" : null,
            "n_siret" : null,
            "type_detablissement" : null,
            "sigle" : null,
            "statut" : null,
            "tutelle" : null,
            "universite" : null,
            "boite_postale" : null,
            "commune_cog" : null,
            "cedex" : null,
            "telephone" : null,
            "arrondissement" : null,
            "departement" : null,
            "academie" : null,
            "region" : null,
            "region_cog" : null,
            "debut_portes_ouvertes" : null,
            "fin_portes_ouvertes" : null,
            "commentaires_portes_ouvertes" : null,
                "lien_site_onisepfr" : null},
 "@type" : "ItemList",
 "itemListElement": [{"code_uai":"0875030V","n_siret":"48391194700016","type_detablissement":"Centre de
 c\u00e9ramique","sigle":"","statut":"priv\u00e9","tutelle":"","universite":"","boite_postale":"","adres
 "commune_cog":"87085","cedex":"","telephone":"05 55 30 08 08","arrondissement":"","departement":"87 - H
 "region_cog":75,"longitude_x":1.2681,"latitude_y":45.8108,"debut_portes_ouvertes":"05\/03\/2021","fin_p
 "commentaires_portes_ouvertes":"de 09h30 \u00e0 16h30","lien_site_onisepfr":"www.onisep.fr\/http\/redir
```

We can see the context command where we put 'null' if we don't want a property.

Important notice: We neglected our ontology and used the uploaded files. However, we are now aware about the fact that we just had to change the URI of our ontology to have the same result.

Fuseki server:

We created a **Fuseki** server. We imported our rdf files for each POI dataset created. This way, the datasets are populated with the **train stations**, **post offices** and **transports** all around Ile-de-France and **schools** in Paris, Bordeaux, Lyon and Lille.


Manage datasets

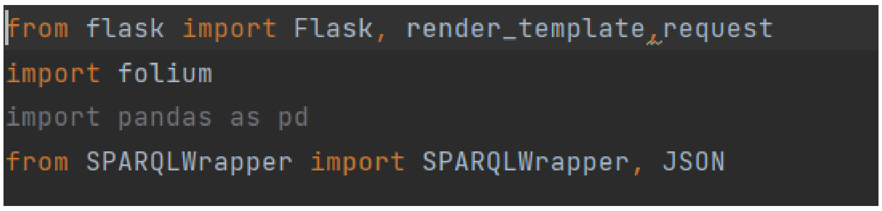
existing datasets | new dataset | tasks

Filter datasets

name	Actions
/EcoleSup	query remove backup add data info
/Gares	query remove backup add data info
/Postes	query remove backup add data info
/Velib	query remove backup add data info

Code:

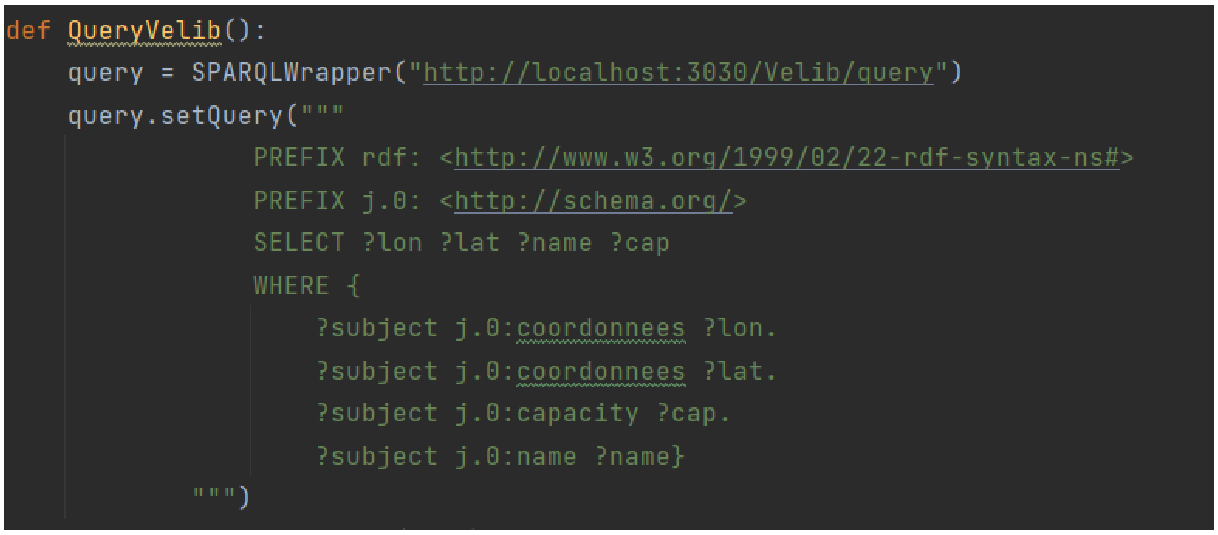
Import packages:


```
from flask import Flask, render_template,request
import folium
import pandas as pd
from SPARQLWrapper import SPARQLWrapper, JSON
```

We need **flask** for the UI, **folium** for the map and **SQPARQLWrapper** to use the queries.

SPARLQ queries:

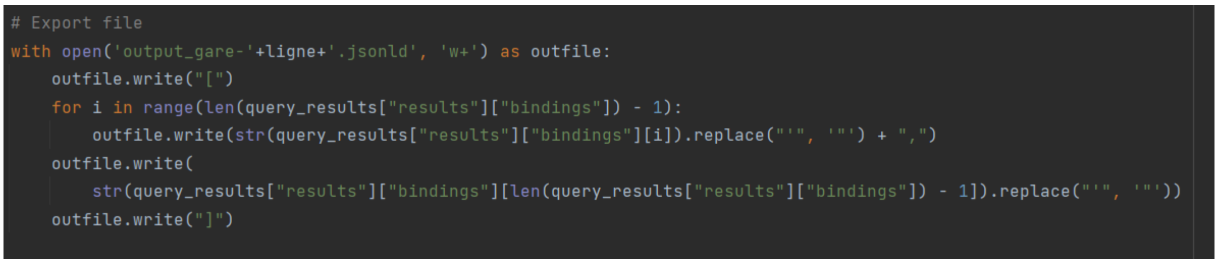
We used simple SPARQL queries that get the essential properties we need to map in the app.



```python
def QueryVelib():
    query = SPARQLWrapper("http://localhost:3030/Velib/query")
    query.setQuery("""
                PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                PREFIX j.0: <http://schema.org/>
                SELECT ?lon ?lat ?name ?cap
                WHERE {
                    ?subject j.0:coordonnees ?lon.
                    ?subject j.0:coordonnees ?lat.
                    ?subject j.0:capacity ?cap.
                    ?subject j.0:name ?name}
            """)
```

This is an example for the Velib. We need the **long** and **lat** for the **coordinates** to show it in the map, the **name**, and the **capacity** (how many bikes it contains).

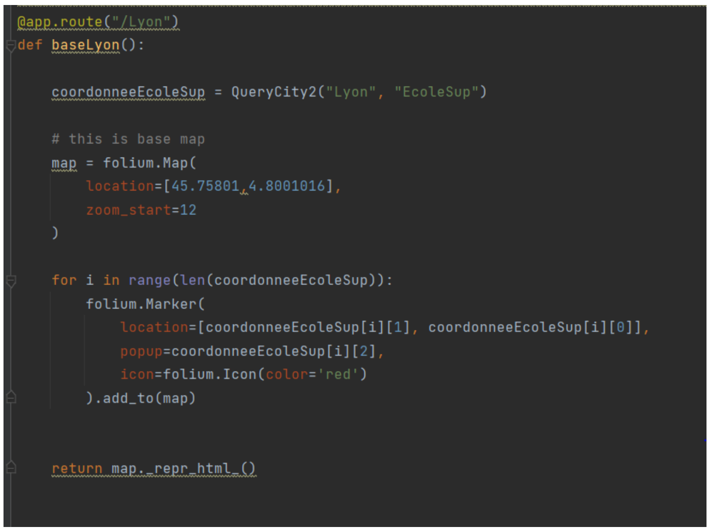
Export file:



```python
# Export file
with open('output_gare-'+ligne+'.jsonld', 'w+') as outfile:
    outfile.write("[")
    for i in range(len(query_results["results"]["bindings"]) - 1):
        outfile.write(str(query_results["results"]["bindings"][i]).replace("'", '"') + ",")
    outfile.write(
        str(query_results["results"]["bindings"][len(query_results["results"]["bindings"]) - 1]).replace("'", '"'))
    outfile.write("]")
```

The result of the query is exported in a **jsonld** file. This is an example for stations using a line name.

Flask:

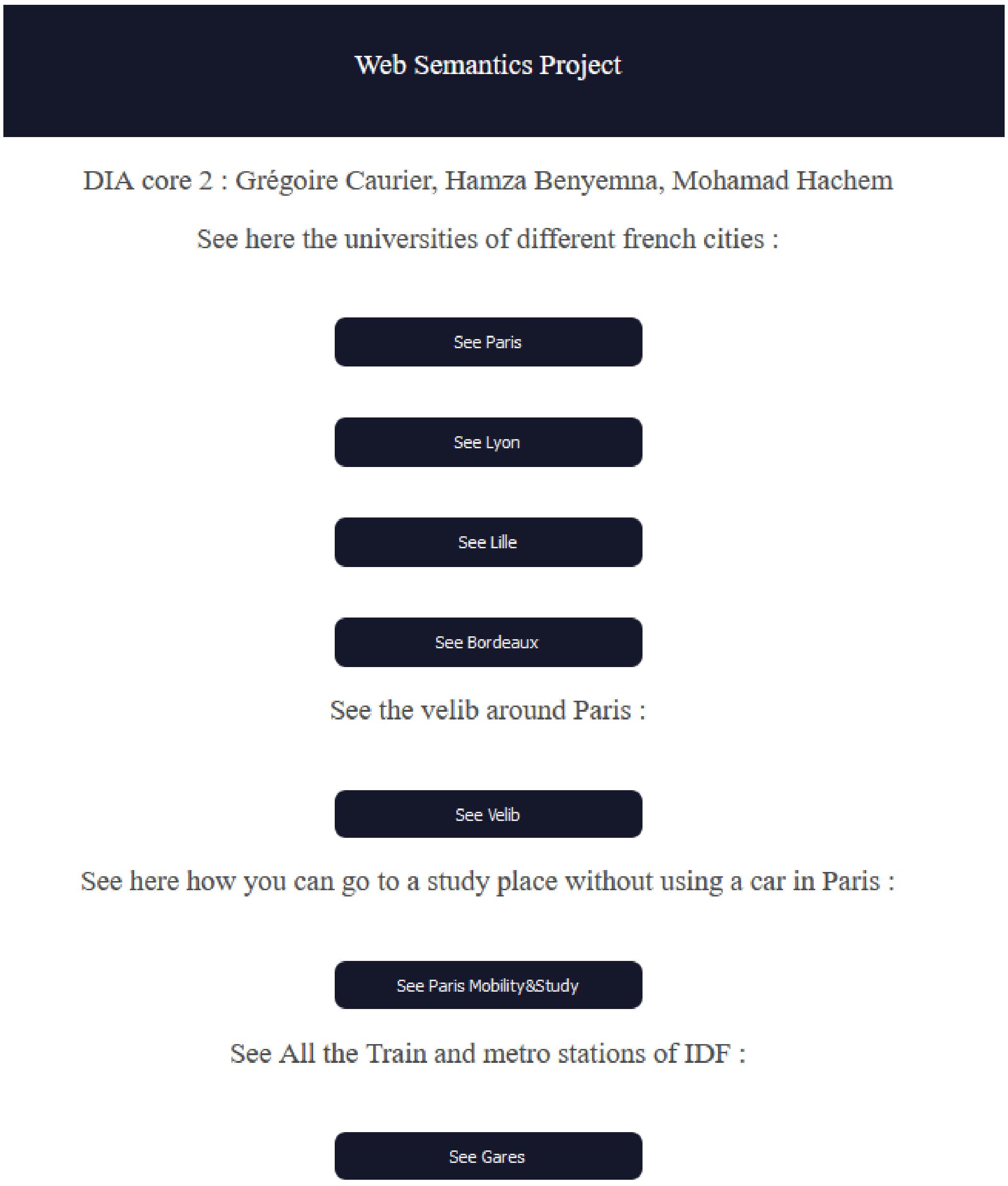


```python
@app.route("/Lyon")
def baseLyon():

    coordonneeEcoleSup = QueryCity2("Lyon", "EcoleSup")

    # this is base map
    map = folium.Map(
        location=[45.75801,4.8001016],
        zoom_start=12
    )

    for i in range(len(coordonneeEcoleSup)):
        folium.Marker(
            location=[coordonneeEcoleSup[i][1], coordonneeEcoleSup[i][0]],
            popup=coordonneeEcoleSup[i][2],
            icon=folium.Icon(color='red')
        ).add_to(map)


    return map._repr_html_()
```

This code allows the connection between our **Fuseki** server and our **html/css** UI.

Overview of the app:

When we run the project and we open the **index.htlm** in the templates folder with **base.css** file, we have this:



Web Semantics Project

DIA core 2 : Grégoire Caurier, Hamza Benyemna, Mohamad Hachem

See here the universities of different french cities :

See the velib around Paris :

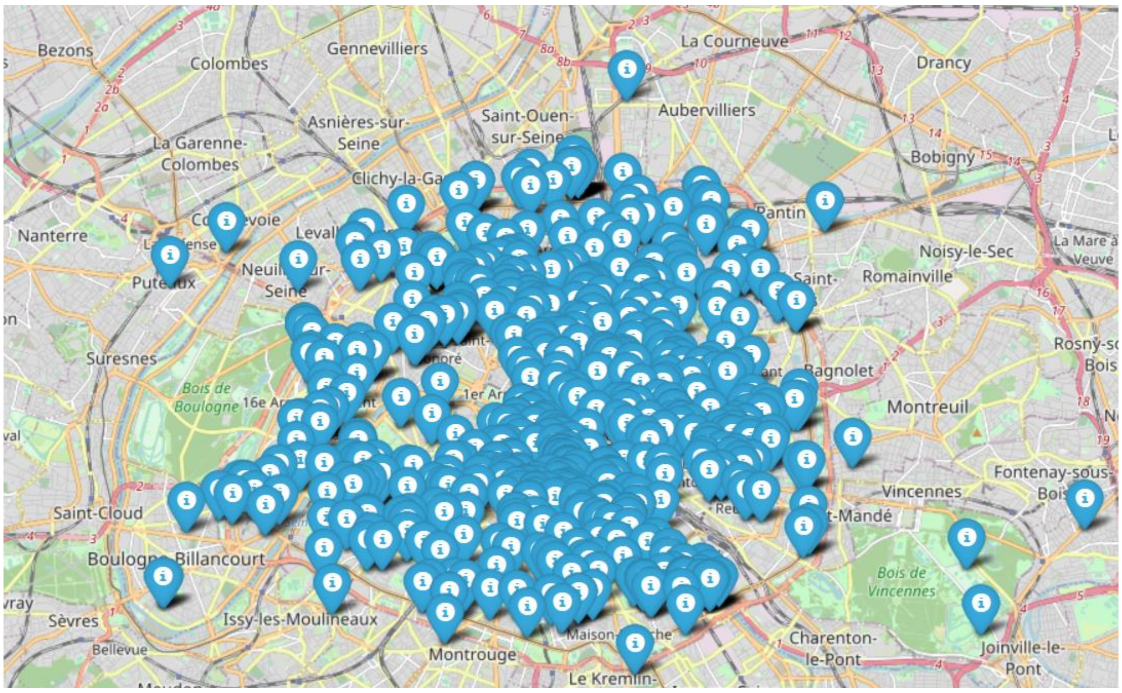
See here how you can go to a study place without using a car in Paris :

See All the Train and metro stations of IDF :

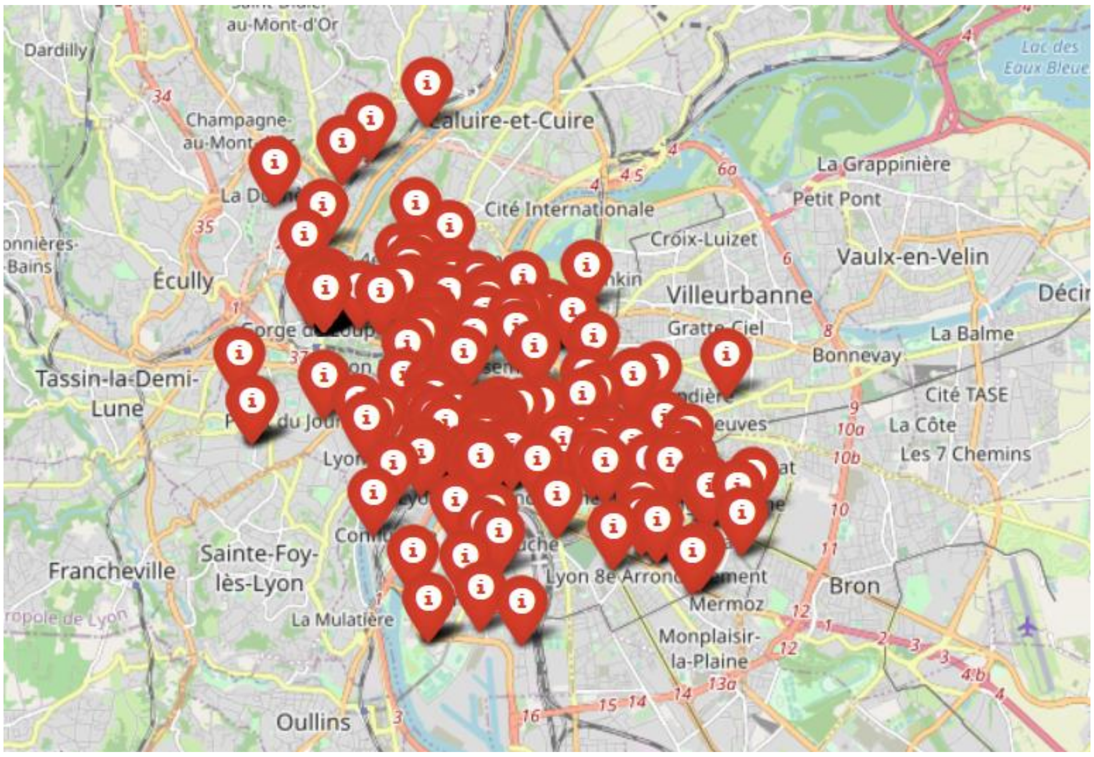
Then; we just have to click on 'See…' whatever we want.

Universities of different cities:

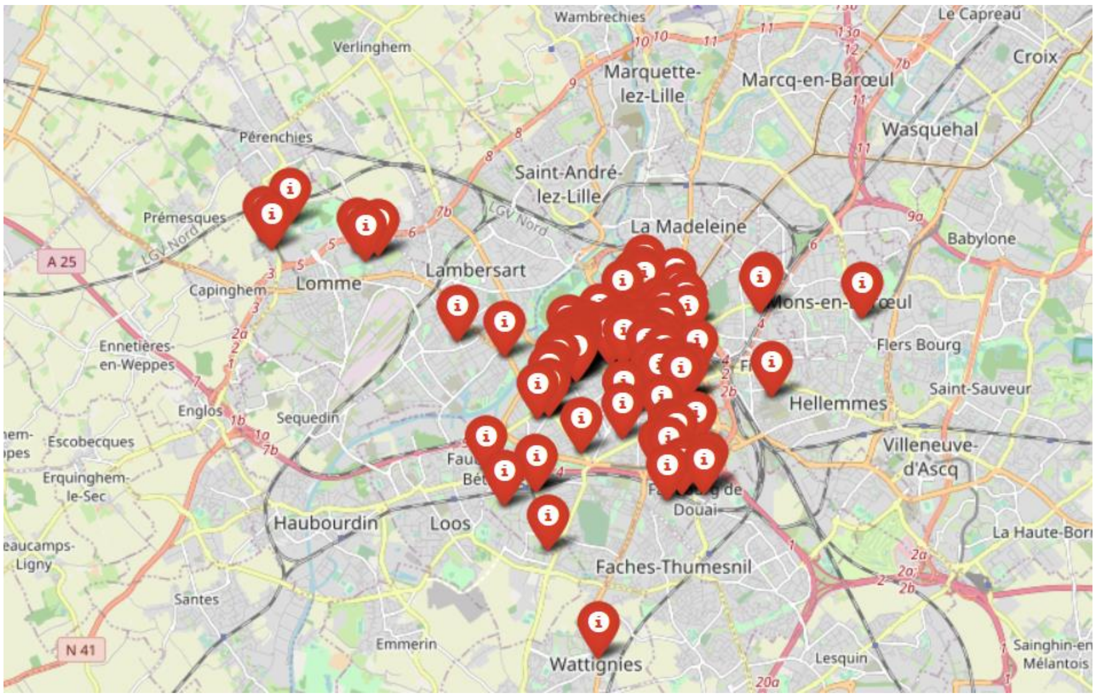
Paris:



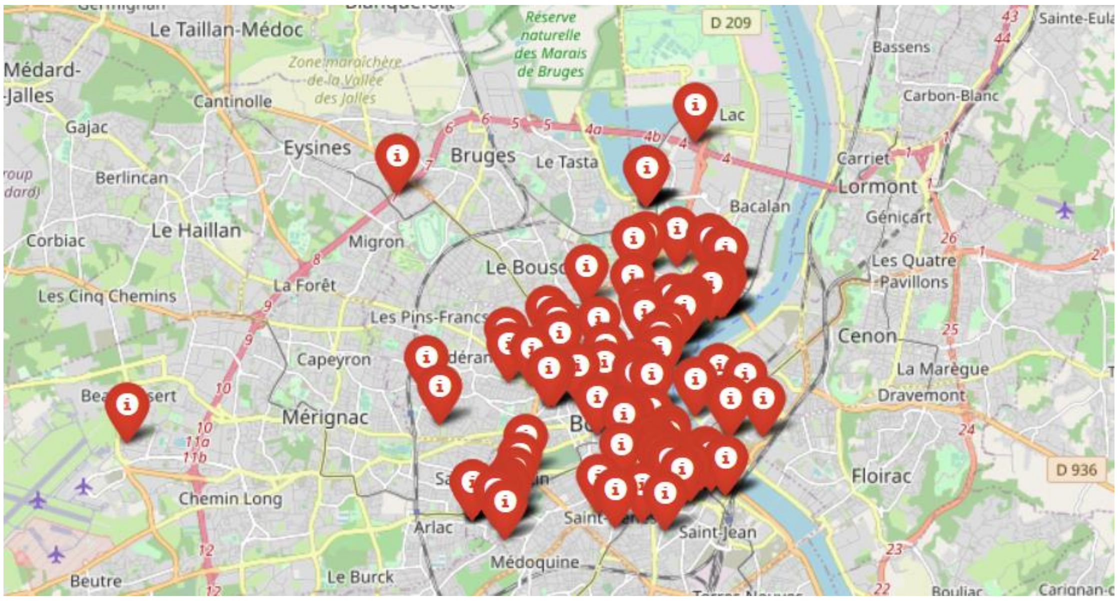
Lyon:



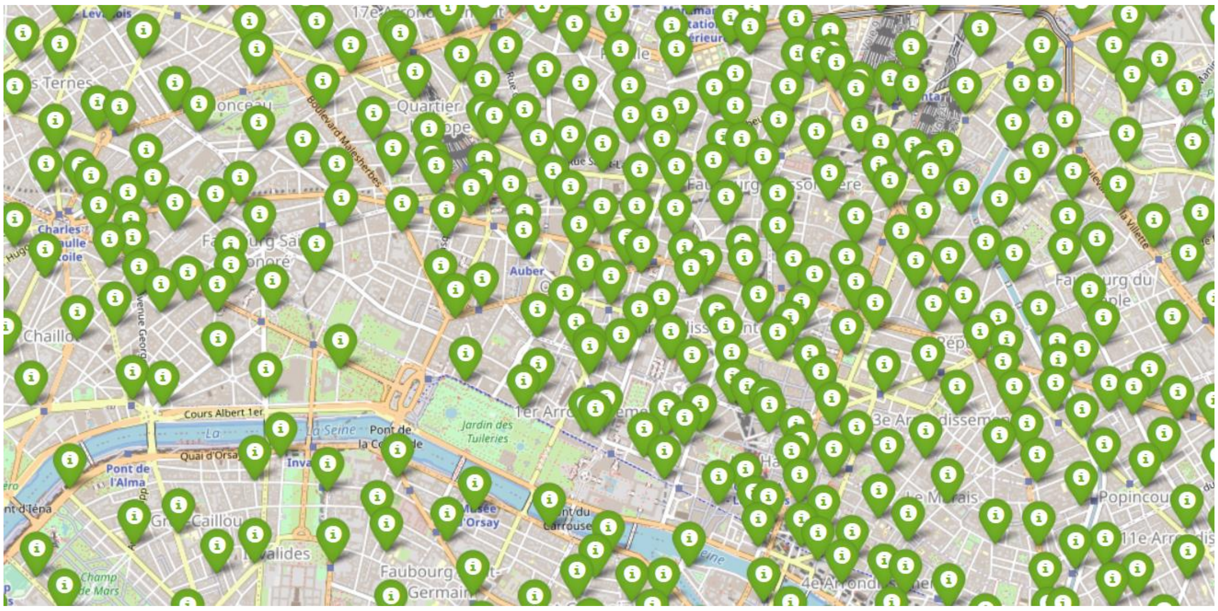
Lille:



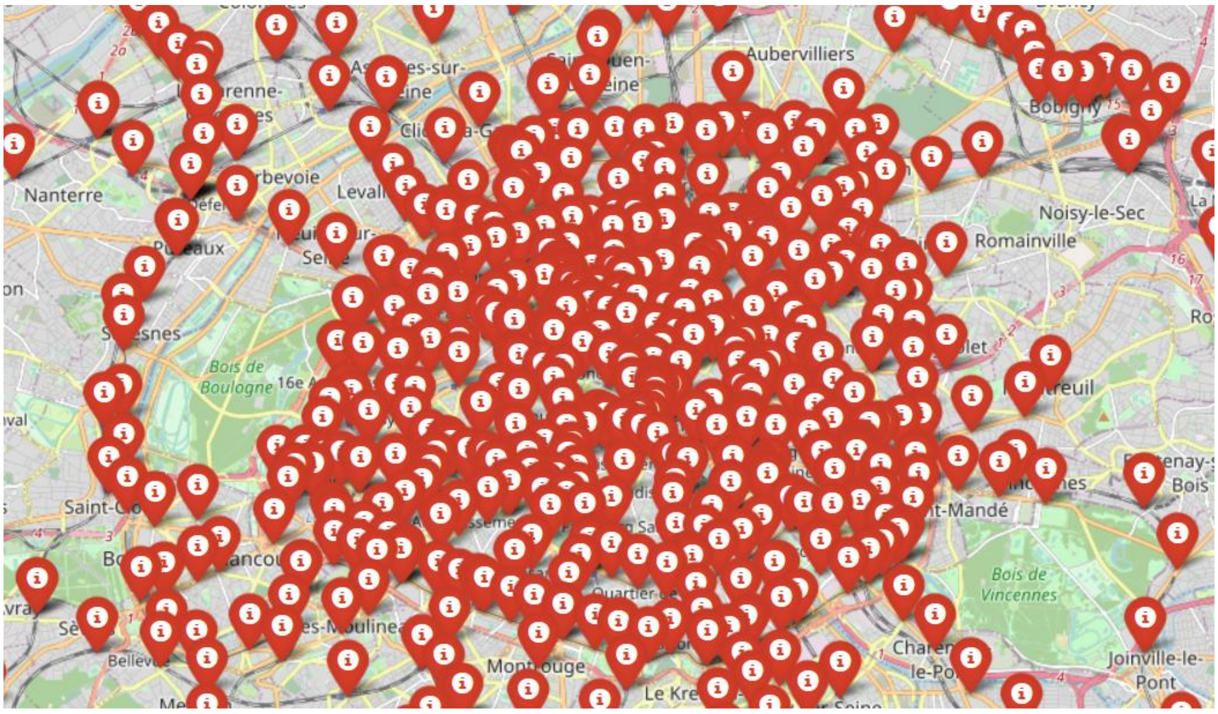
Bordeaux:



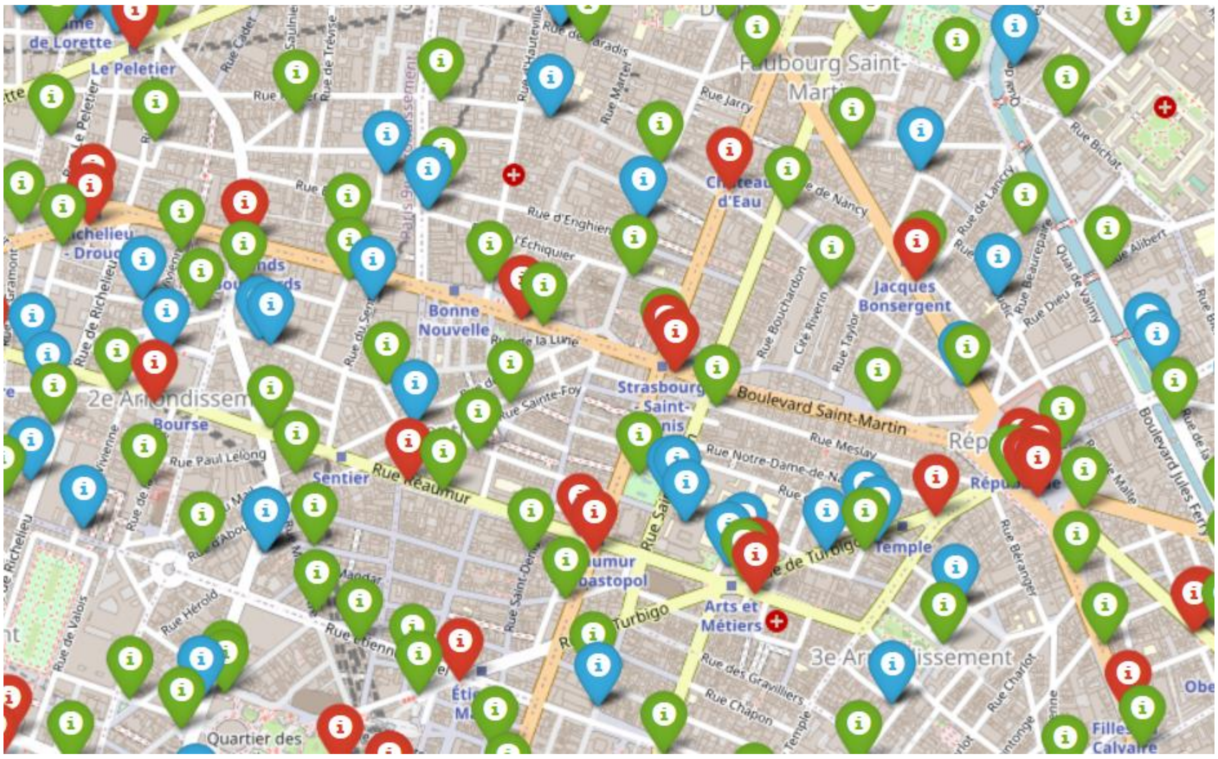
Velib:



Stations:



All stations, Velib and universities of Paris:

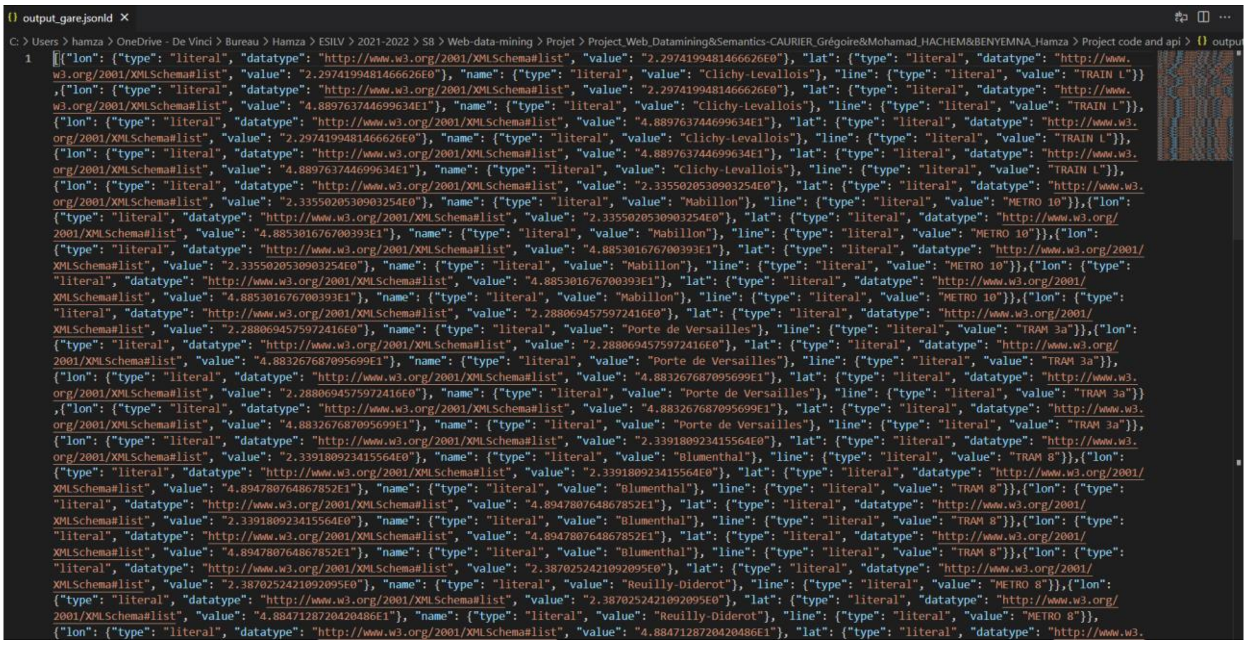


This helps a **traveller** to have an idea of transports and Velib in order to organise a **trip**.

Export:

output_gare.jsonld

A **jsonld** file is created with the results of the query in the **Project.py** folder named 'Project code and api'.

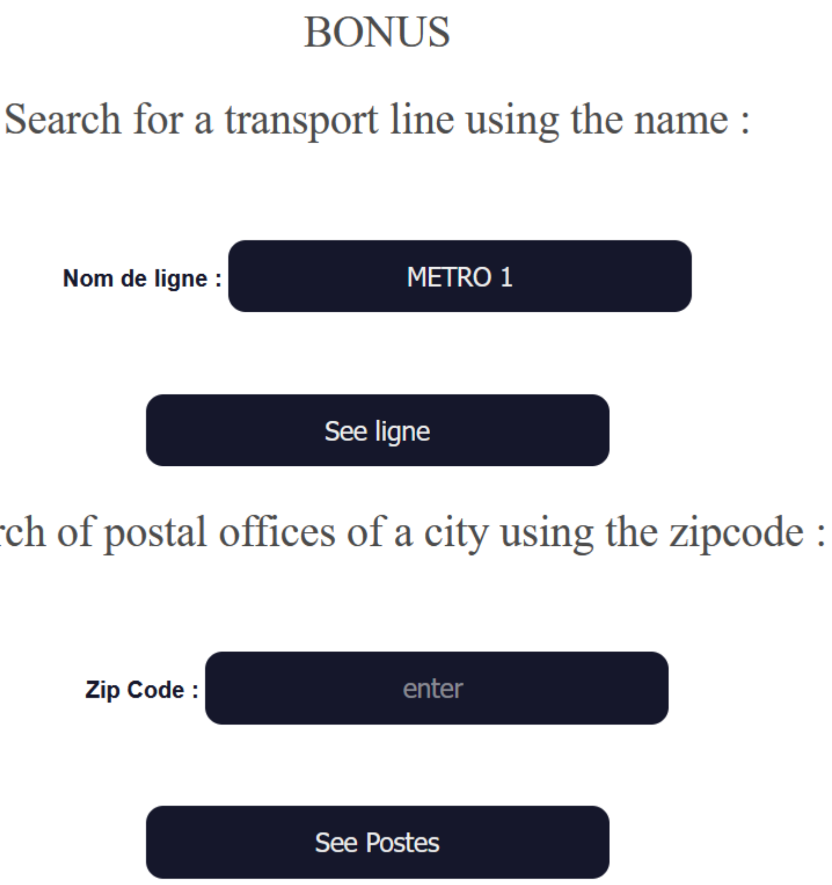

output_gare.jsonld ×

C: > Users > hamza > OneDrive - De Vinci > Bureau > Hamza > ESILV > 2021-2022 > S8 > Web-data-mining > Projet > Project_Web_Datamining&Semantics-CAURIER_Grégoire&Mohamad_HACHEM&BENYEMNA_Hamza > Project code and api > output

```
[{"lon": {"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "2.2974199481466626E0"}, "lat": {"type": "literal", "datatype": "http://www.
w3.org/2001/XMLSchema#list", "value": "2.2974199481466626E0"}, "name": {"type": "literal", "value": "Clichy-Levallois"}, "line": {"type": "literal", "value": "TRAIN L"}}
,{"lon": {"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "2.2974199481466626E0"}, "lat": {"type": "literal", "datatype": "http://www.
w3.org/2001/XMLSchema#list", "value": "4.889763744699634E1"}, "name": {"type": "literal", "value": "Clichy-Levallois"}, "line": {"type": "literal", "value": "TRAIN L"}},
{"lon": {"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "4.889763744699634E1"}, "lat": {"type": "literal", "datatype": "http://www.w3.
org/2001/XMLSchema#list", "value": "2.2974199481466626E0"}, "name": {"type": "literal", "value": "Clichy-Levallois"}, "line": {"type": "literal", "value": "TRAIN L"}},
{"lon": {"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "4.889763744699634E1"}, "lat": {"type": "literal", "datatype": "http://www.w3.
org/2001/XMLSchema#list", "value": "4.889763744699634E1"}, "name": {"type": "literal", "value": "Clichy-Levallois"}, "line": {"type": "literal", "value": "TRAIN L"}},
{"lon": {"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "2.3355020530903254E0"}, "lat": {"type": "literal", "datatype": "http://www.w3.
org/2001/XMLSchema#list", "value": "2.3355020530903254E0"}, "name": {"type": "literal", "value": "Mabillon"}, "line": {"type": "literal", "value": "METRO 10"}},{"lon":
{"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "2.3355020530903254E0"}, "lat": {"type": "literal", "datatype": "http://www.w3.org/
2001/XMLSchema#list", "value": "4.885301676700393E1"}, "name": {"type": "literal", "value": "Mabillon"}, "line": {"type": "literal", "value": "METRO 10"}},{"lon":
{"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "4.885301676700393E1"}, "lat": {"type": "literal", "datatype": "http://www.w3.org/2001/
XMLSchema#list", "value": "2.3355020530903254E0"}, "name": {"type": "literal", "value": "Mabillon"}, "line": {"type": "literal", "value": "METRO 10"}},{"lon": {"type":
"literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "4.885301676700393E1"}, "lat": {"type": "literal", "datatype": "http://www.w3.org/2001/
XMLSchema#list", "value": "4.885301676700393E1"}, "name": {"type": "literal", "value": "Mabillon"}, "line": {"type": "literal", "value": "METRO 10"}},{"lon": {"type":
"literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "2.2880694575972416E0"}, "lat": {"type": "literal", "datatype": "http://www.w3.org/2001/
XMLSchema#list", "value": "2.2880694575972416E0"}, "name": {"type": "literal", "value": "Porte de Versailles"}, "line": {"type": "literal", "value": "TRAM 3a"}},{"lon":
{"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "2.2880694575972416E0"}, "lat": {"type": "literal", "datatype": "http://www.w3.org/
2001/XMLSchema#list", "value": "4.883267687095699E1"}, "name": {"type": "literal", "value": "Porte de Versailles"}, "line": {"type": "literal", "value": "TRAM 3a"}},
{"lon": {"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "4.883267687095699E1"}, "lat": {"type": "literal", "datatype": "http://www.w3.
org/2001/XMLSchema#list", "value": "2.2880694575972416E0"}, "name": {"type": "literal", "value": "Porte de Versailles"}, "line": {"type": "literal", "value": "TRAM 3a"}}
,{"lon": {"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "4.883267687095699E1"}, "lat": {"type": "literal", "datatype": "http://www.w3.
org/2001/XMLSchema#list", "value": "4.883267687095699E1"}, "name": {"type": "literal", "value": "Porte de Versailles"}, "line": {"type": "literal", "value": "TRAM 3a"}},
{"lon": {"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "2.339180923415564E0"}, "lat": {"type": "literal", "datatype": "http://www.w3.
org/2001/XMLSchema#list", "value": "2.339180923415564E0"}, "name": {"type": "literal", "value": "Blumenthal"}, "line": {"type": "literal", "value": "TRAM 8"}},{"lon":
{"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "2.339180923415564E0"}, "lat": {"type": "literal", "datatype": "http://www.w3.org/2001/
XMLSchema#list", "value": "4.894780764867852E1"}, "name": {"type": "literal", "value": "Blumenthal"}, "line": {"type": "literal", "value": "TRAM 8"}},{"lon": {"type":
"literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "4.894780764867852E1"}, "lat": {"type": "literal", "datatype": "http://www.w3.org/2001/
XMLSchema#list", "value": "2.339180923415564E0"}, "name": {"type": "literal", "value": "Blumenthal"}, "line": {"type": "literal", "value": "TRAM 8"}},{"lon": {"type":
"literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "4.894780764867852E1"}, "lat": {"type": "literal", "datatype": "http://www.w3.org/2001/
XMLSchema#list", "value": "4.894780764867852E1"}, "name": {"type": "literal", "value": "Blumenthal"}, "line": {"type": "literal", "value": "TRAM 8"}},{"lon": {"type":
"literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "2.3870252421092095E0"}, "lat": {"type": "literal", "datatype": "http://www.w3.org/2001/
XMLSchema#list", "value": "2.3870252421092095E0"}, "name": {"type": "literal", "value": "Reuilly-Diderot"}, "line": {"type": "literal", "value": "METRO 8"}},{"lon":
{"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "2.3870252421092095E0"}, "lat": {"type": "literal", "datatype": "http://www.w3.org/
2001/XMLSchema#list", "value": "4.8847128720420486E1"}, "name": {"type": "literal", "value": "Reuilly-Diderot"}, "line": {"type": "literal", "value": "METRO 8"}},
{"lon": {"type": "literal", "datatype": "http://www.w3.org/2001/XMLSchema#list", "value": "4.8847128720420486E1"}, "lat": {"type": "literal", "datatype": "http://www.w3.
```

This export file looks like this with a list of all the **stations** in this example.

Bonus:


BONUS

Search for a transport line using the name :

Nom de ligne : METRO 1

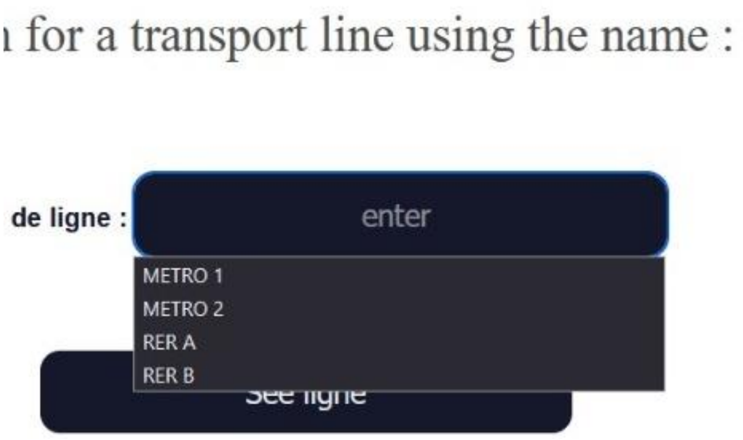
Search of postal offices of a city using the zipcode :

Zip Code : enter

Bonus notice: dynamic queries, the export also is dynamic because it exports on file per registered input (line for stations and zip code for post offices)

We have dynamic functions that shows us the stations of a line and post offices that we want. There is also a **historic** stored of the entered inputs.

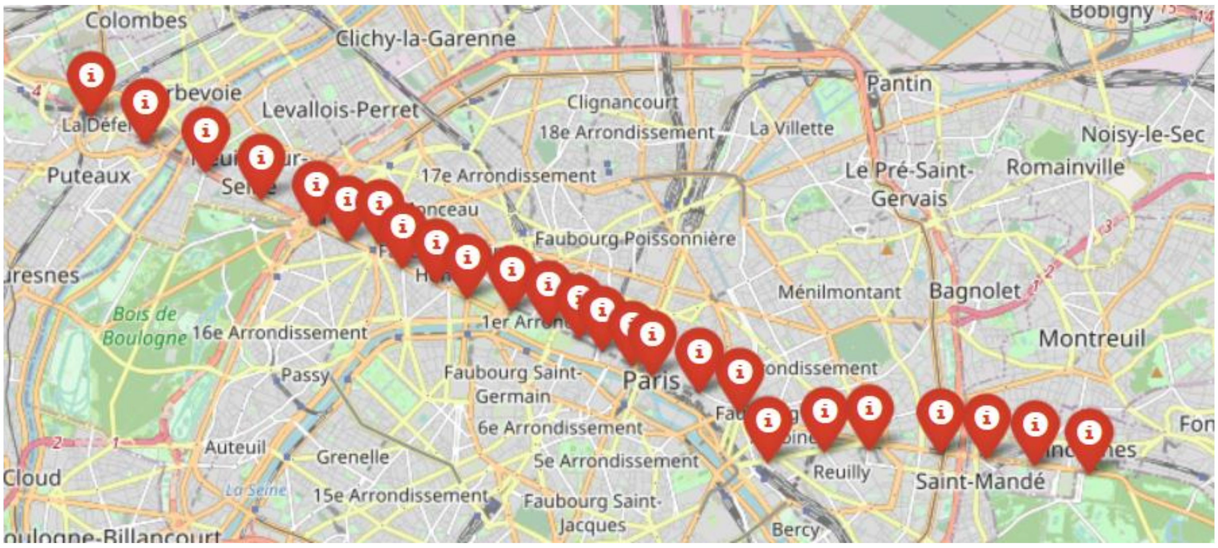
Stations using a line name:


ı for a transport line using the name :

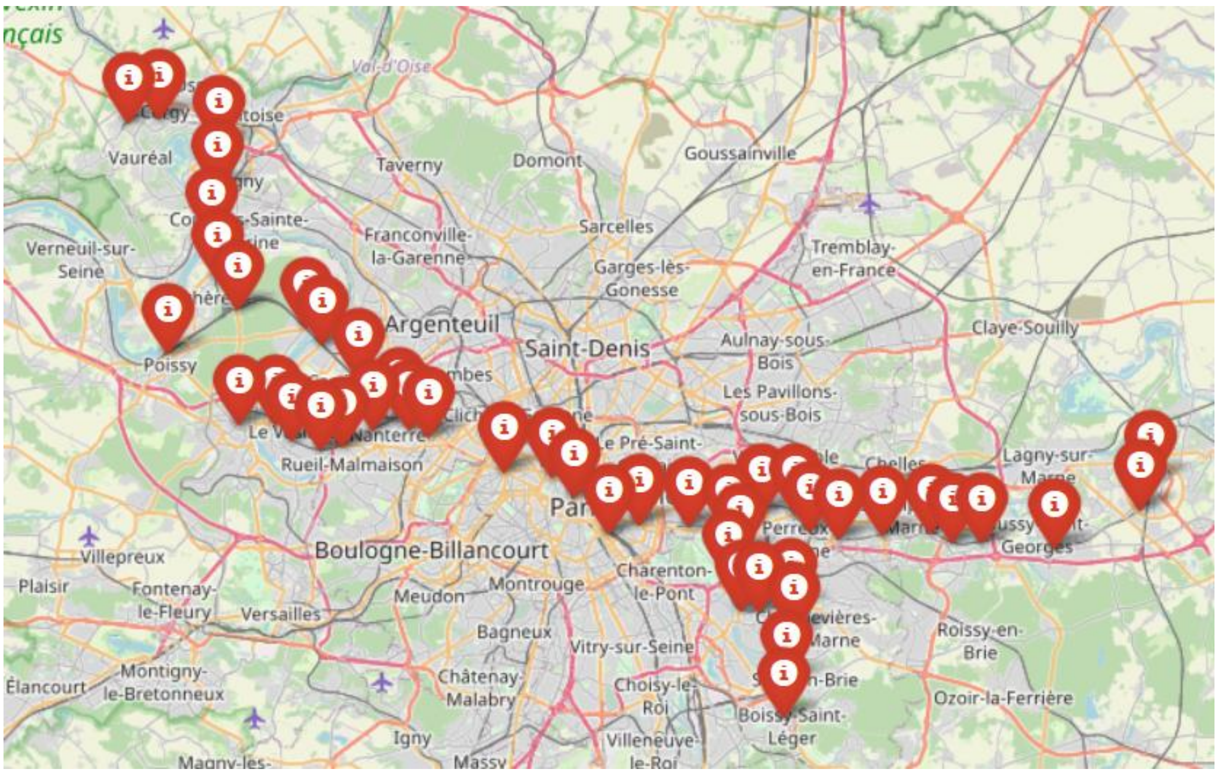
de ligne : enter

For example, we will ask for 'METRO 1' and 'RER A'.

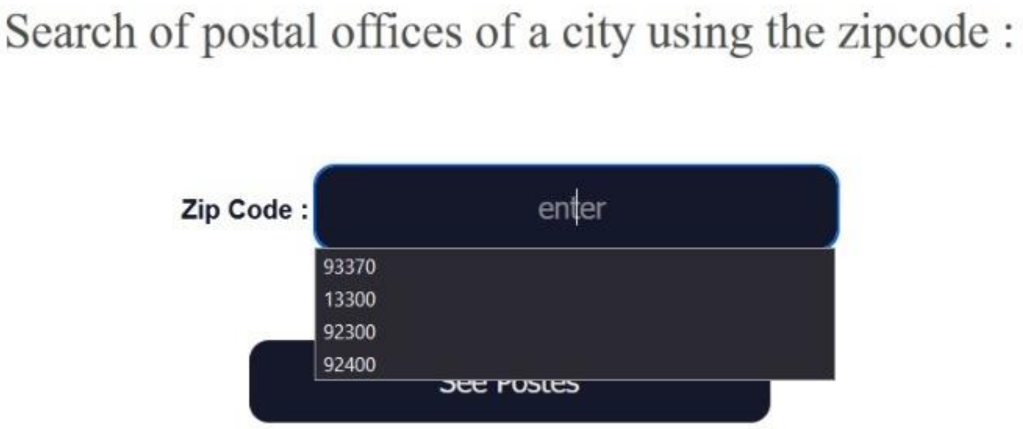
METRO 1:



RER A:

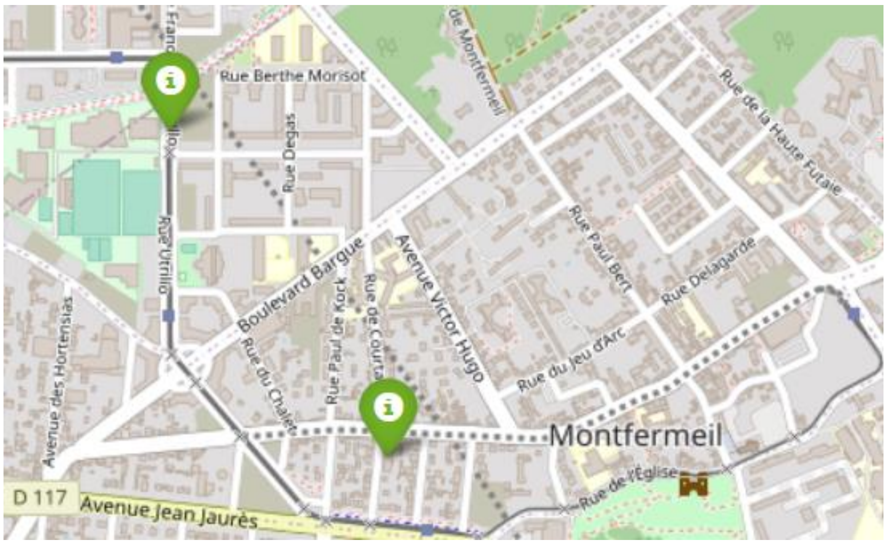


Post offices using a zip code:

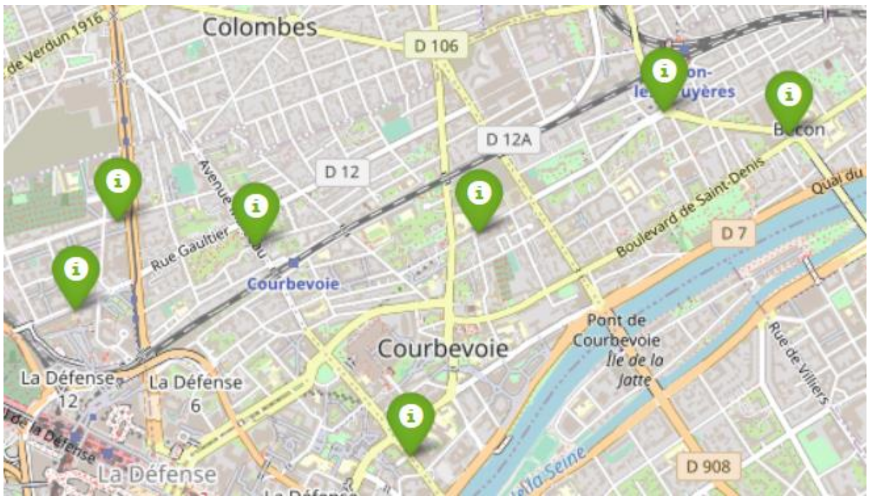

Search of postal offices of a city using the zipcode :

For example, we will ask for '93370 and '92400.

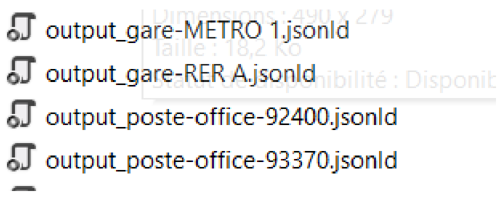
93370:



92400:



Dynamic exported files:



output_gare-METRO 1.jsonld
output_gare-RER A.jsonld
output_poste-office-92400.jsonld
output_poste-office-93370.jsonld

For each query, we have a different file given an input.