Une image contenant texte

Description générée automatiquement



Semantics & Web datamining Report

**A4 - S8**

**ESILV**

<https://github.com/NicolasCarval/Web-DataMining-Project/>

*PINCET Bruno*

*CARVAL Nicolas*

*BUQUET Sebastien*

1. **Introduction**

We produced a web page that shows POIs such as Train stations and museums on a map (you can do advanced research on criterias such as: city, frequentation, Wi-Fi access...). It also tells you about weather in different cities, and compute train trips between 2 train stations of your choice.

In the end you have the option to download RDF triples in turtle formats or JSON-LD representing the trips computed.

1. **Technologies used**

* Creation of the ontology: protege
* Individuals: python with the rdflib library
* Triple data store: Apache jena fuzeki
* Web application (HTML, CSS, JS):
  + Query: localhost API fuzeki
  + Dynamic Data: API SNCF, API OpenWeather
  + Map: Leafletjs

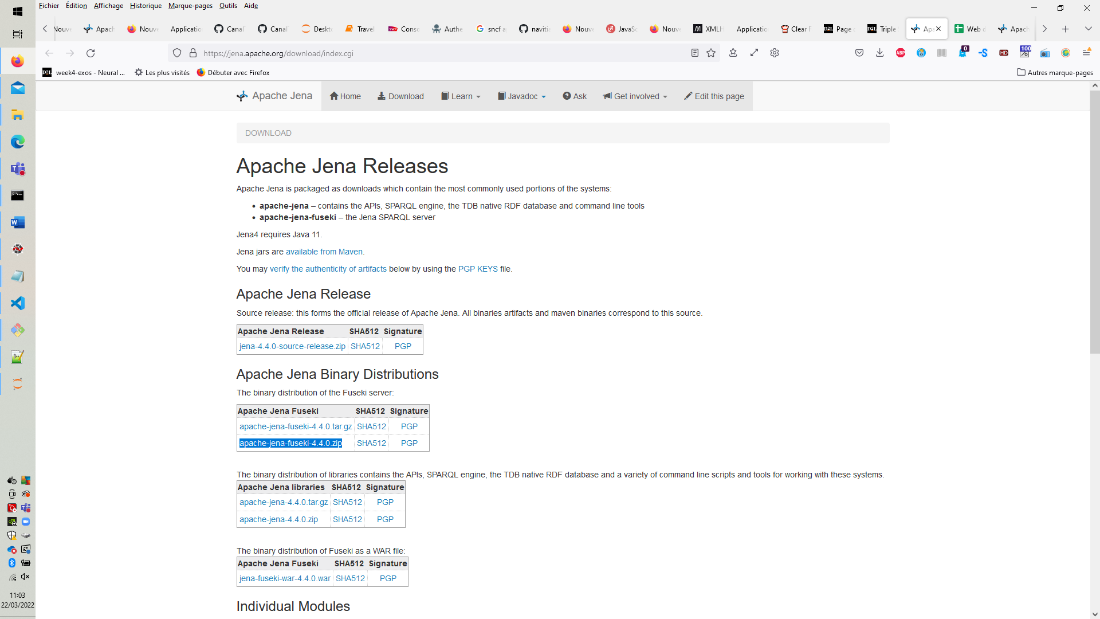
1. **How to install**

* Fork GitHub repository

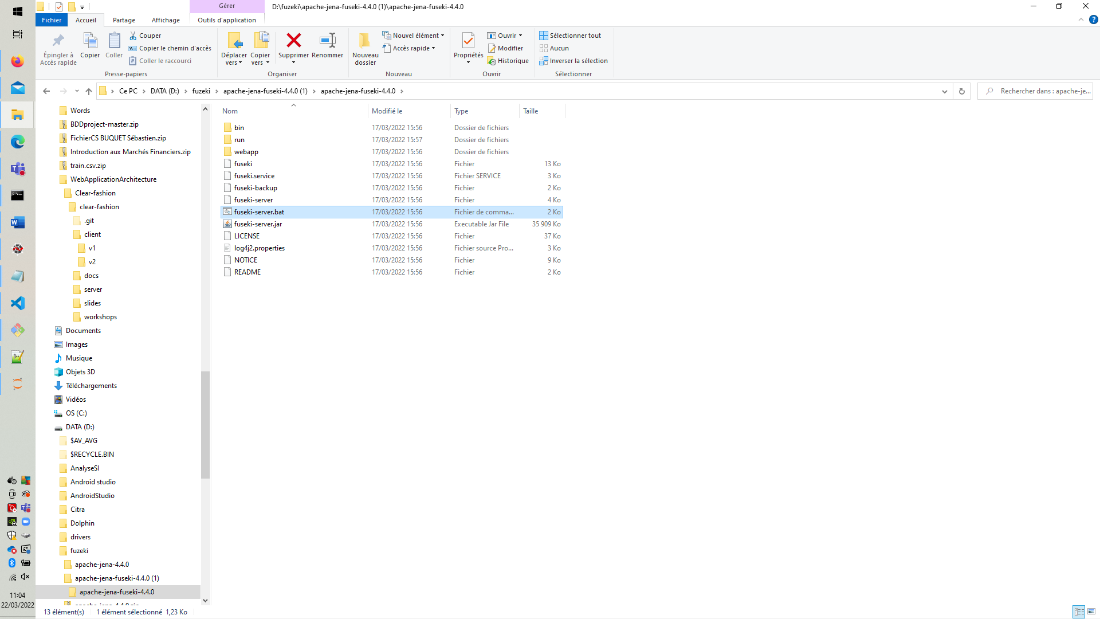
<https://github.com/NicolasCarval/Web-DataMining-Project/>

* Setup the triple store

To install our file, you first need to download Fuseki form this link: <https://jena.apache.org/download/index.cgi/>

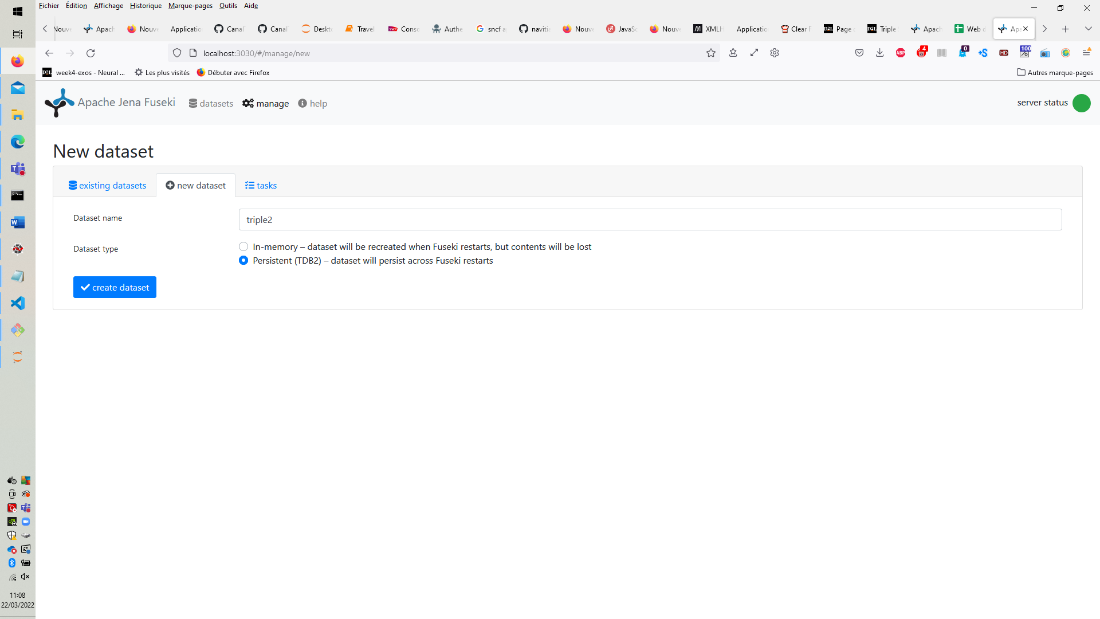


You then need to unzip the downloaded file and launch fuzeki-server.bat



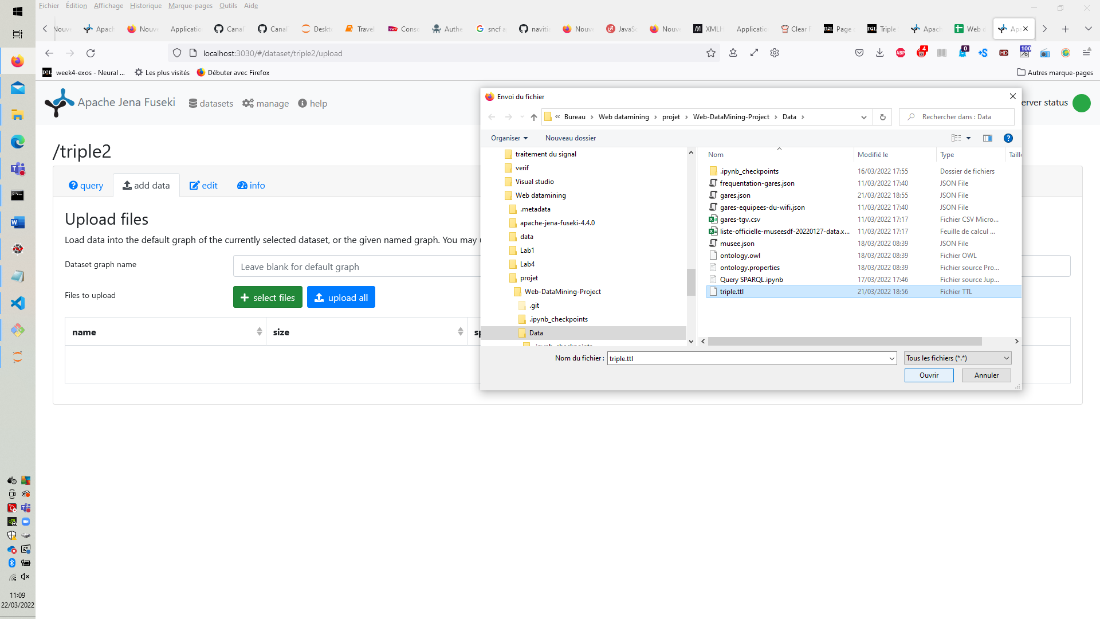
Launch a browser and enter <http://localhost:3030/>

Create a database and call it triple2



Click on add data and choose the file triple.ttl from the zip that was given to you.

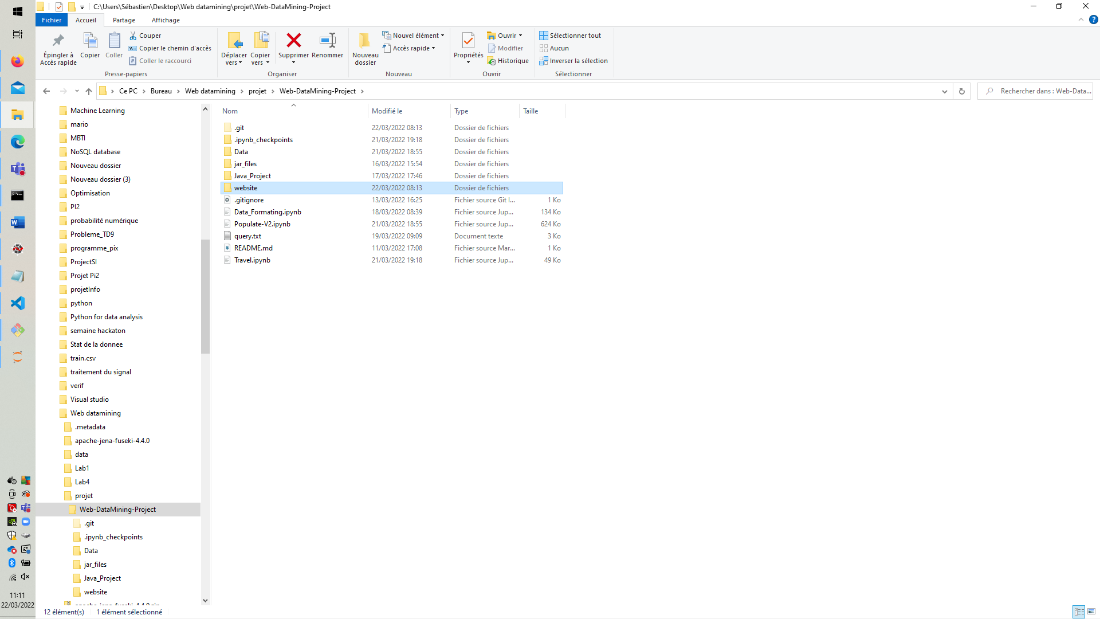
Then click on upload now.



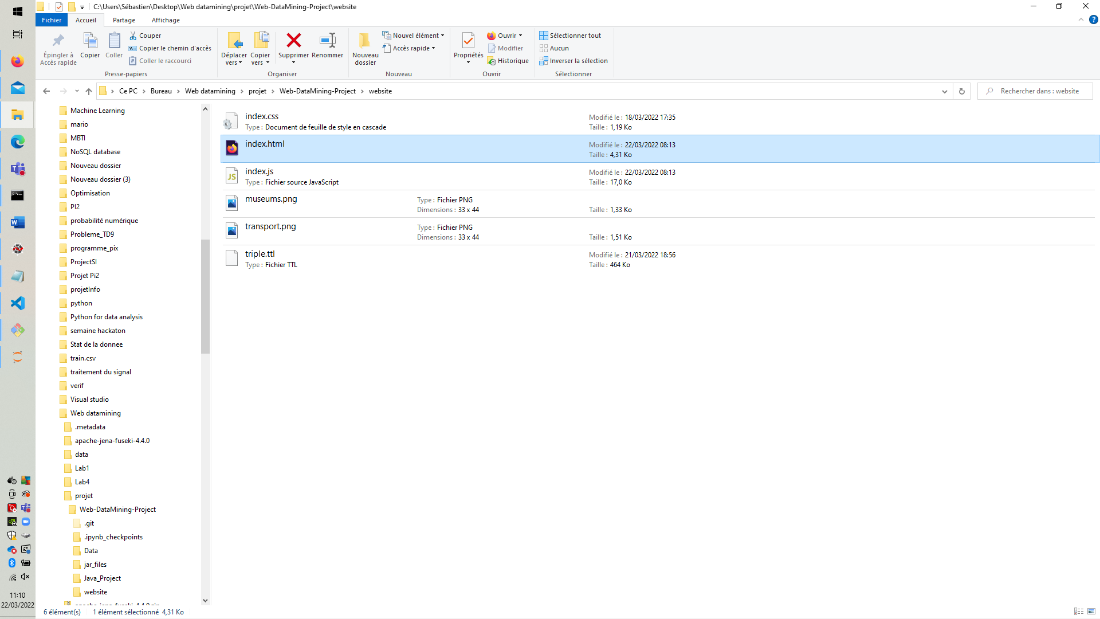
The triple store is now setup

* **Launch**

In our project folder go the website folder

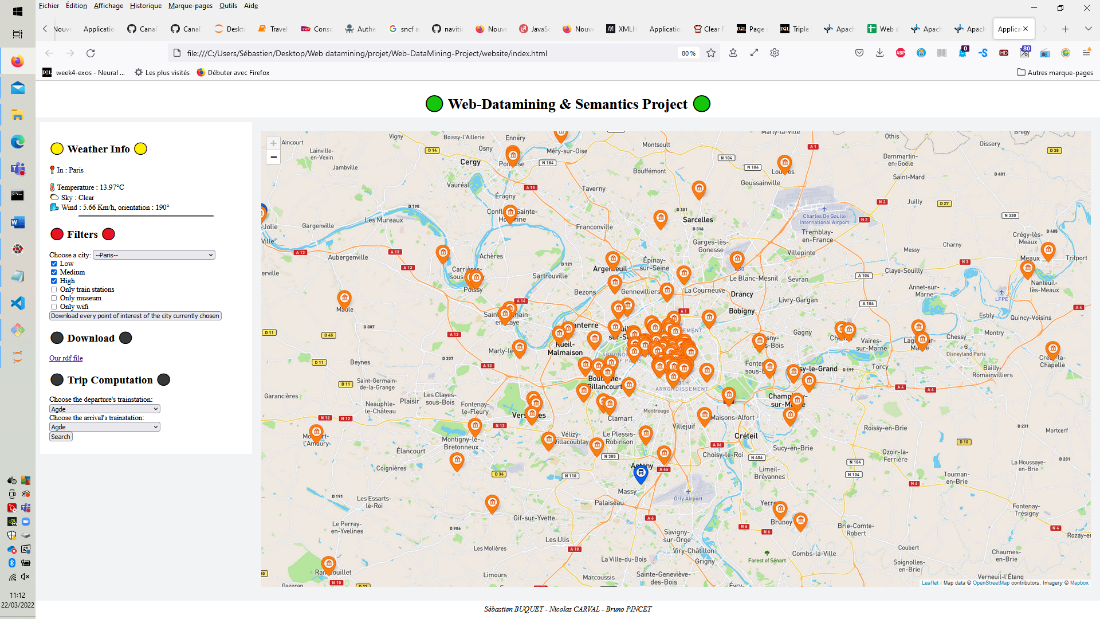


Double click on the index.html



If the map is not well displayed, please dezoom until you have the filters on the left and the map on the right of the page.

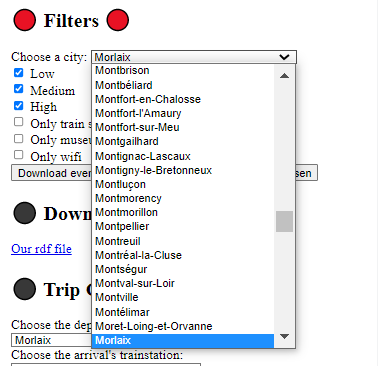
1. **How to use**



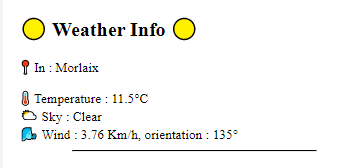
Our web application displays information from our triple store on the map. We choose to display the museums and train stations.

We then added filters:

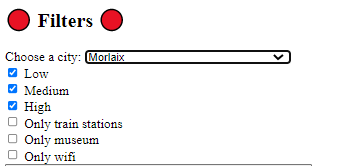
* The user can choose the city he wants to see from a list



* Choosing a city also displays the meteo of the city.

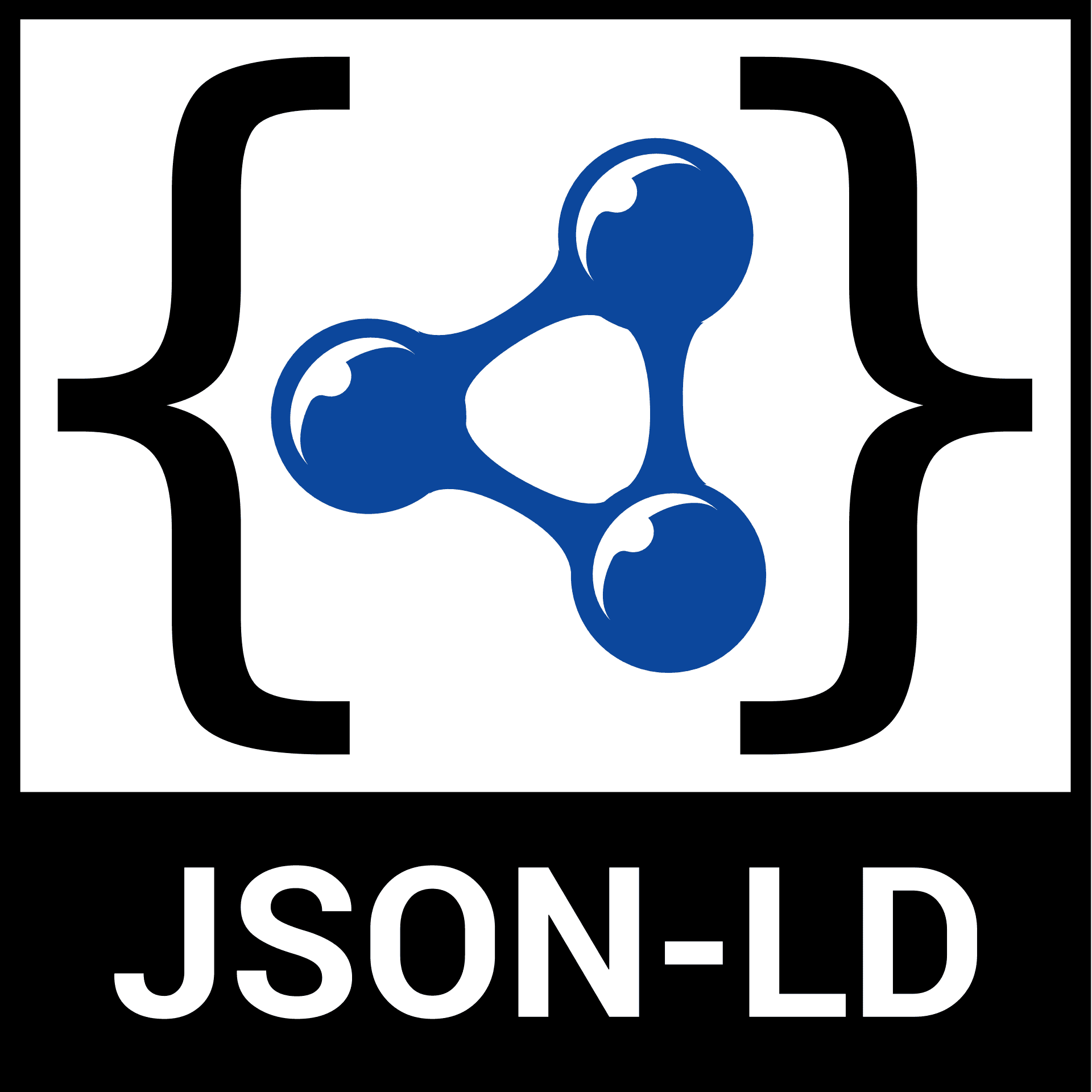
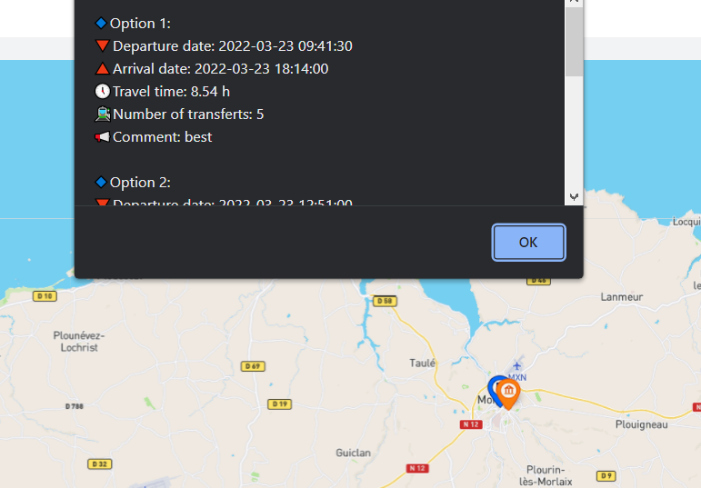


* Choose the availability of the Wi-Fi and the number of users as criteria for the selection process.

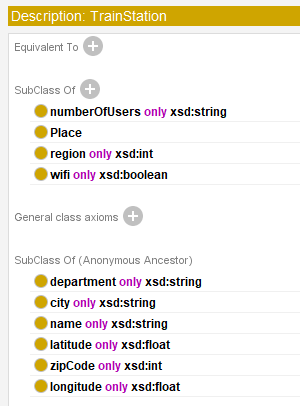
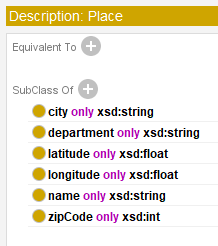
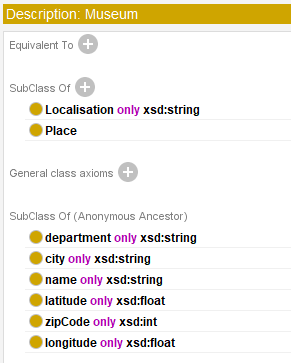


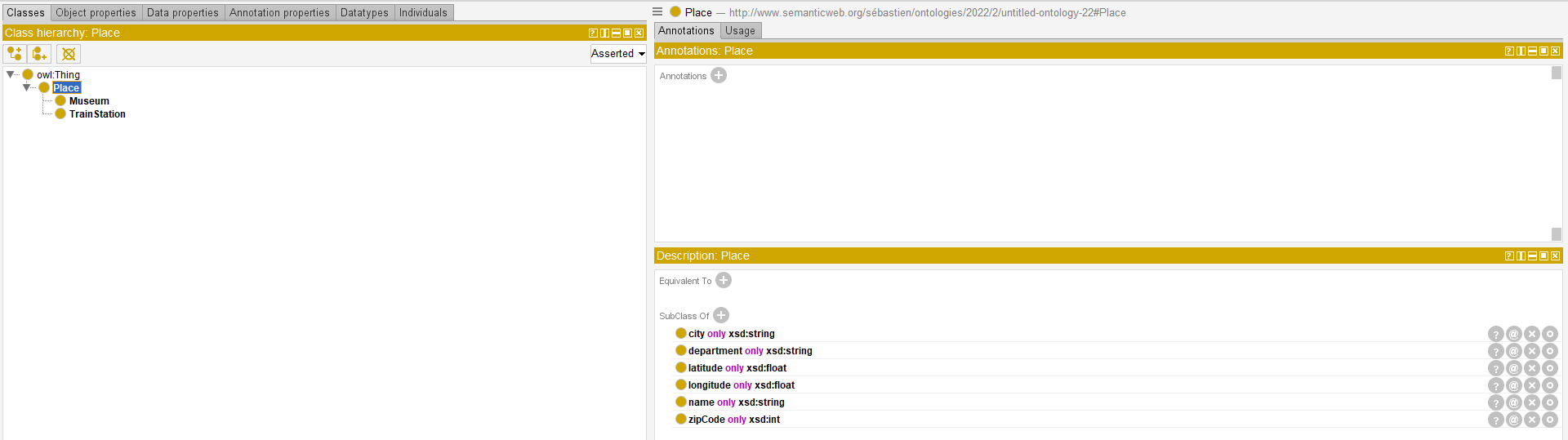
The user can also download all the information we have on the city and the points of interest that are inside it. All the information comes from our triple store in RDF.

In the download section, by clicking on “Our RDF file”, the user can download all the data stocked in the triple store.

In the trip computation section, the user can choose two train stations and we compute the possible journeys. We also give the user the result in JSON-LD format to make it possible for him to study the information later according to schema.org



1. **Ontology:**

****

1. **Additional queries:**

**Prefix used:**

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ns1: <http://www.semanticweb.org/sebastien/ontologies/2022/2/untitled-ontology-22#>

**#List the instances of the geolocated POI**

SELECT distinct ?name ?longitude ?latitude

WHERE {​​​​​​

?x rdf:type ?type.

?type rdfs:subClassOf\* ns1:Place.

?x ns1:name ?name.

?x ns1:longitude ?longitude.

?x ns1:latitude ?latitude.

}​​​​​​

**#List the name of all train station. For each one, display its city.**

SELECT distinct ?name ?city WHERE {​​​​​​

?trainStation rdf:type ns1:TrainStation.

?trainStation ns1:name ?name.

?trainStation ns1:city ?city.

}​​​​​​

**#List the name of train station with wifi.**

SELECT ?name

WHERE

{​​​

?s rdf:type ns1:TrainStation .

?s ns1:wifi ?x.

?s ns1:name ?name

FILTER ( ?x="Oui")

}​​​

**#2 Optional Graph Patterns query**

SELECT ?name ?lon ?lat ?nbu ?wifi

WHERE

{​​​

?s rdf:type ?type.

?type rdfs:subClassOf\* ns1:Place.

?s ns1:longitude ?lon.

?s ns1:latitude ?lat.

?s ns1:city ?city.

?s ns1:name ?name.

OPTIONAL {​​​?s ns1:numberOfUsers ?nbu. }​​​.

OPTIONAL {​​​?s ns1:wifi ?wifi .}​​​.

}​​​

**#A query that contains at least 2 alternatives and conjunctions , museum and Train station with wifi**

SELECT ?name

WHERE{

{ ?x rdf:type ns1:TrainStation.

?x ns1:name ?name.

?x ns1:wifi "Oui"}

UNION{

?y rdf:type ns1:Museum.

?y ns1:name ?name.

}

}

**#A query that contains a CONSTRUCT query form**

CONSTRUCT {​​​

?x ?p ?o.

}​​​

WHERE {​​​

?x rdf:type ?type.

?type rdfs:subClassOf\* ns1:Place.

?x ?p ?o.

}​​​

**#A query that contains an ASK query form**

Test l'existence d'un musée dans Paris

ASK{

?s rdf:type ns1:Museum .

?s ns1:city "Paris".

}

**# A query that contains a DESCRIBE query for**

DESCRIBE ?place

WHERE {​​​ ?place a ns1:TrainStation ;}​​​

1. **Self-assessment**

The self-assessment is based on the previous explanations and some additional information.

**Part 1 (2.5/4 pt) :classes and properties (1.5/2 pt), restrictions and defined classes (0.5/1pt),Extern ontology reuse (0.5/1 pt)**

We realise two classes that depends from on class Place:

* Museum
* Train Station

Concerning the dynamic travel data that are created when the user asks for them, we used the Schema.org ontology.

We create several data properties: name, city, zip code, Wi-Fi, number of users, longitude, latitude, adresse.

To export the journeys options between train stations in JSON-LD we use format according to the schema.org ontology.

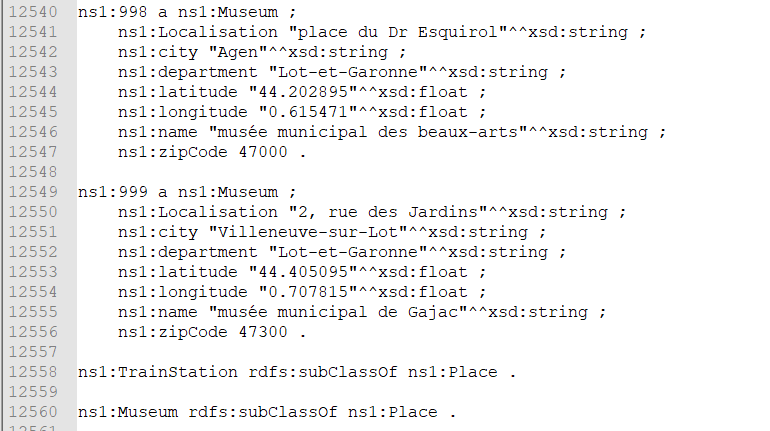
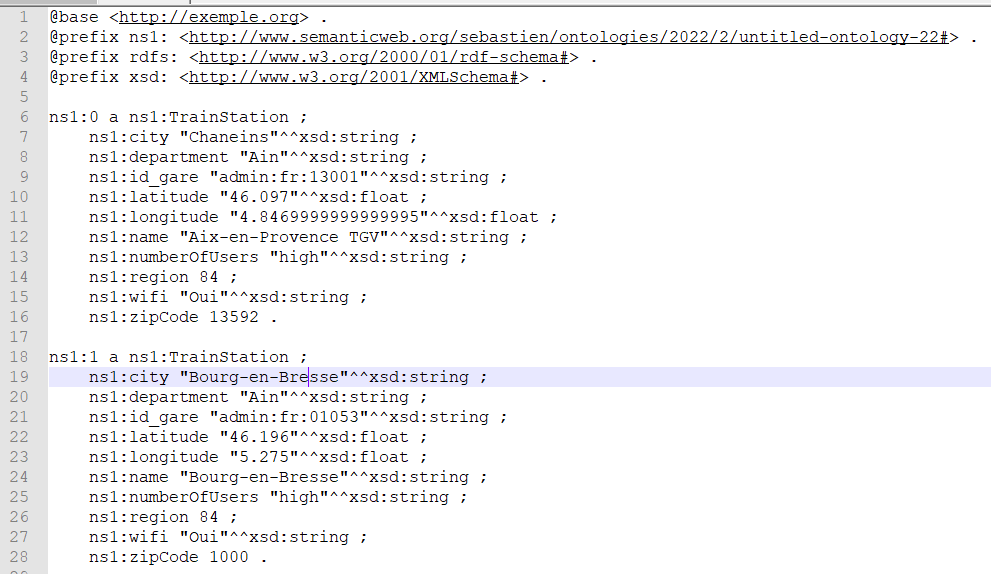
**Part 2 (3.75/4 pts): individuals (0.75/1pt), sources (2/2pt: more than 2 sources, 1pt: one source only), entities: more than one entity (class) (1/1pts);**

We created a dataset of more than 1 000 individuals for train stations and museums.

We used the SNCF dataset for the train stations list, and we combined it with other datasets to add information on Wi-Fi access and traffic. We used the government dataset for museums and the SNCF API for the trips. We have five sources of information.

We have two entities: Train stations and Museums.

To populate and create our individuals we use RDFlib in python. The code used is available in the file Populate-V2.ipynb in the folder Data\_formating.



**Part 3 (2.5/4 points): SPARQL (2.5/3 pts), SWRL (0/1 pts)**

We realise several queries (construct, describe, ask) that we stored in a file called SPARQL\_Query.txt but we only use a part of these queries in our web application.

**Part 4 (5.5/8pts): Entity search (POI, travellers, trip) ---> (1 pts each: 1.5/3pts); Advanced functionality (could trip computation, or any other advanced functionality such trip computation or traveller history) -->(1.5/2pts); any other information associated to POI such as weather forecasting (or another entity) --> 1.5/2pts; result export in RDF or JSON-LD (1/1pt)**

In our web application we have:

* entity search for city (static data)
* trip computation (dynamic data)
* weather forecasting for the different city where there is a POI with an API call (dynamic data)
* export our data in RDF for the POI of a city and export the journeys options between train stations in JSON-LD format according to the schema.org ontology. (Static and dynamic data).

**Bonus (6 pts): triple store (2/2pts); GUI (1.5/2pts), RDFA (0/1 pt) , Realtime dynamic data update (1.5/2pt), very good presentation (1 pt) etc …. 1 pt for each mentioned extension**

We use the triple store Apache Jena Fuseki and Realtime dynamic data update with the weather.

We also have a very nice presentation with a web application that shows the POI on a map and allows the user to act on the map.