Ecole des Mines

SEMANTIC WEB PROJECT

Lyon City Guide Query System Reinforced by Semantic Web Technology

Submitted by:
Poulomi NANDY
Rediet TADESSE

Supervisor:
Prof. Antoine ZIMMERMANN

January 3, 2021

Contents

1 Introduction

Semantic Web is the extension of the World Wide Web,in which information is given well-defined meaning, better enabling computers and people to work in cooperation.Semantic web technologies are becoming more famous and popular in the field of web development , IoT fields which provide concise solutions for existing problems of the internet and communication world. Not only this Semantic Web will also have popularity in the feild of Machine Learning. Todays Document based web pages will be difficult for the machines to understand and to find out the correct meaning.Therefore Semantic Web is gaining popularity as this provides structured data to the web so that with human beings it helps machines as well to find out the correct meaning.

This report explains how we utilize semantic web technologies to solve a problem which copes with day to day lives of current society. This project is discussing a particular city guide domain which talks about **hospitals, SNCF trains, metros, buses, trams, bicycles, weather, museums**. In our project we have considered heterogenous data. Initially, we will focus on the static data then we will explain about the dynamic data and finally the wikidata which we have queried and displayed in our project.

The main objective of our project is to consider few kinds of spatial data.Here, we have focused on Lyon city in France and tried to show all relevant transports,hospital for emergency. In the dynamic part we have focused on the real time data like weather and availability of bicycle. Suppose if a user wants to ride a bicycle then of-course the person needs to see if any bicycle is available in the bicycle station and the current weather. Is the weather feasible to ride a bicycle or not.The last part we have focused on the wikidata. We have queried wikidata for lyon city museums and tried to display that in our web page. Primarily, all the data we have used in our project is in RDF model. The data are getting queried from a triple store and fetched in our User Interface.The information are shown in the form of List.

Evidently, our project is divided into two parts. First, we have modelled the data based on our ontology and saved it in a triple store. Secondly, displaying the data in our web page. In order to achieve this goal we have used SPARQL queries . With semantic web technologies we have also used some traditional web development technologies in our project. The second part of the project is again divided into 2 parts: Front end i.e User Interface and Back end i.e Server Side.

2 Data Used

2.1 Static data:

All the data files are integrated within the project and no need to download the files explicitly.

First **Hospital Data** in the form of JSON. We have displayed spatial information i.e Latitude and Longitude, type of hospital,contact number of the hospital and the address of the hospital. The data source:
`https://www.data.gouv.fr/en/datasets/les-etablissements-hospitaliers-franciliens-idf/#_`

Second **SNCF Data** in the form of CSV. We have displayed spatial information of the stations,name of the stations,escalator available or not, elevator available or not,arrival time of the SNCF train and departure time.
`https://ressources.data.sncf.com/explore/dataset/sncf-ter-gtfs/table/`

Third **Bicycle Data:** in the form of JSON. Here for bicycle data we have fetched the information by the API call and then modeled the data. Here we have displayed only the spatial information , name of the bicycle station and the capacity of the bicycle. No real time data is fetch in the 1st web page.The URL from which the data is being fetched is below:
`https://download.data.grandlyon.com/ws/grandlyon/pvo_patrimoine_voirie.pvostationvelov/all.json?maxfeatures=100&start=1`

Fourth **Bus, Metro, Trams Data:** in the form of CSV. The information displayed are the spatial information,bus-metro-tram stop, bus-metro-tram number. The CSV has been converted and cleaned from the api:
`https://download.data.grandlyon.com/wfs/rdata?SERVICE=WFS&VERSION=2.0.0&outputformat=GEOJSON&maxfeatures=30&request=GetFeature&typename=tcl_sytral.tclarret`

2.2 Dynamic Data

In this part, we have considered. The real time data of Bicycle and Weather both are JSON data stored in a triple store.
For **Bicycle** we have already the co-ordinates in the static page. So, in here we have displayed the Station name and available bicycle in that station.The available bicycle is the real time data which we are fetching from the API and then storing in our triple store.The URL for bicycle is:
`https://download.data.grandlyon.com/wfs/rdata?SERVICE=WFS&VERSION=1.1.0&outputformat=GEOJSON&request=GetFeature&typename=jcd_jcdecaux.jcdvelov&SRSNAME=urn:ogc:def:crs:EPSG::4171`

For **Weather Data** we have designed the fetching of data with the number of available bicycle.We have displayed temperature, pressure,humidity,wind-speed.The url for the real time weather data is:
`http://api.openweathermap.org/data/2.5/weather?APPID=6eaa88893a7b68dde346b5c0ed4c980f`

2.3 Wiki data

In this part, as our basic ideology was to design a website for a particular city guide. We have considered to query the museum data from wikipedia.

3 Technologies Used

3.1 Semantic Web Technologies:

Basically this application uses Semantic Web technologies starting from extracting open data to displaying them on the website. We use Protege to build our OWL ontology for the domain of lyon city guide as turtle file as we have learnt during our course. Apache Jena is using to create and access RDF triples with model. Apache Jena Fuseki is a server which uses SPARQL which we have done already in our course practical session. We use Fuseki server to persist the data set and SPARQL to query it this is project specific. We use RDFa when representing data on the website as we have learnt again from the course. In our ontology we have taken help of blank node so that new updated information can be added.

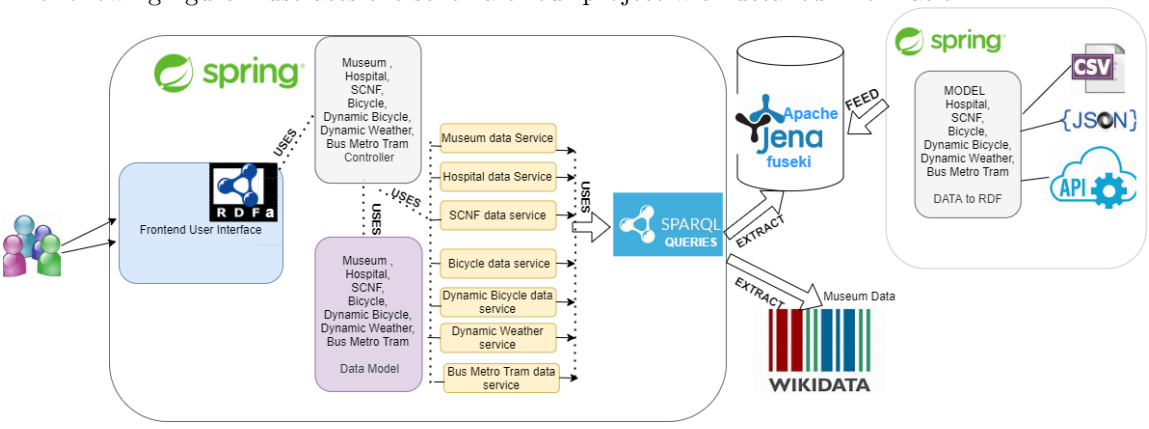
3.2 Web Development Technologies:

Since the system is web based we are using web development frameworks and technologies such as SpringBoot, HTML, Maven, AJAX, jQuery and BootStrap all the mentioned technologies are taught in our course curriculum.

4 System Design

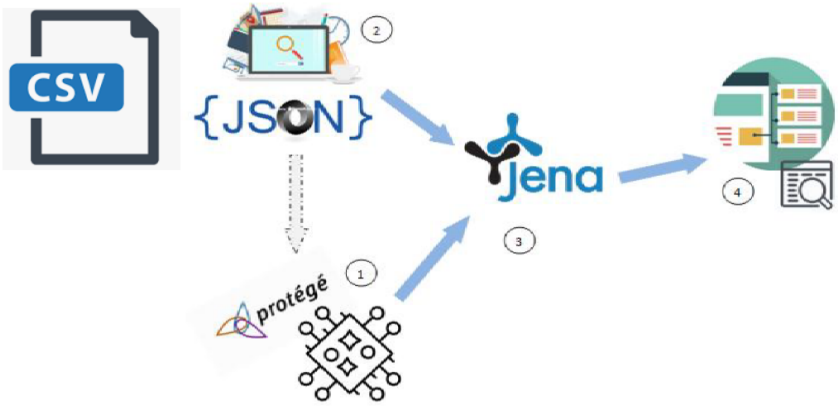
4.1 General schema of the Project:

The following figure illustrates the schema of our project with detailed information



4.2 Basic architecture of the project:

Following figure illustrates the basic architecture of the project. Flow is depicted in arrows with numbers which described below:



Item (1) existing ontologies and ontology we created in Protege by analyzing JSON,CSV data of website.

Item (2) the data of static and dynamic web pages.

Item (3) the RDF model is saving in Fuseki server which is generated using (1), (2).

Item (4) the website visualizing the data from Fuseki server When we design the whole project, it consists of three major parts, which will be described in this section.

Design the Model Structure: One of the most important parts of this project is to model the scenario. We checked the format and content of data we are going to use such as real time and static data.
Extract data and generate the model: Then we decided how to extract static and dynamic data. After extracting data, RDF triplets are generated according to model we designed.
Visualizing the data: We design this as Website and REST API by providing search options over the data set.

5 Planning of Realization

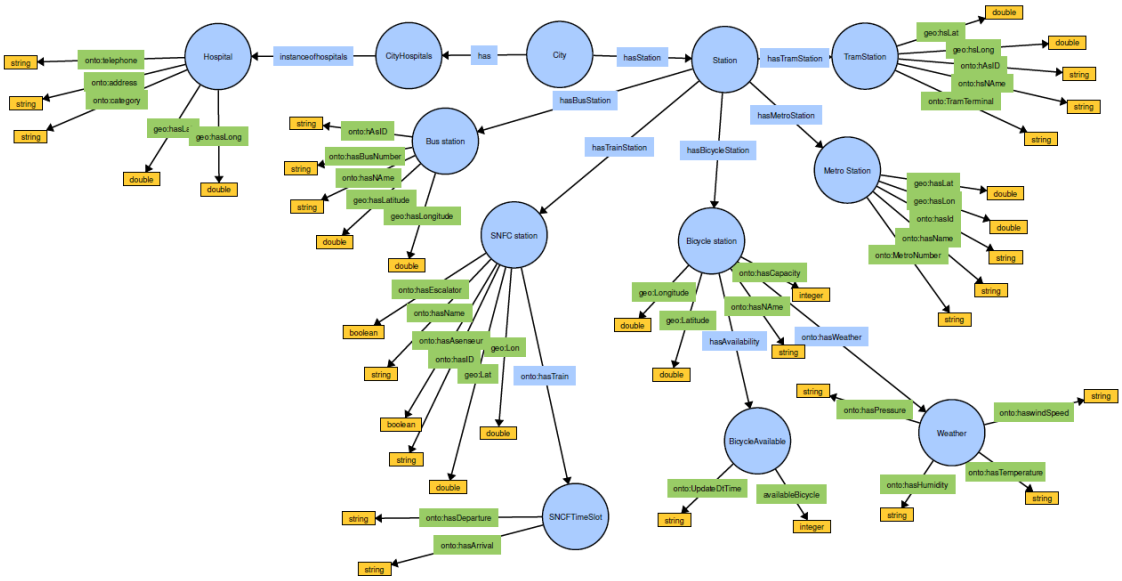
We have two iterations.
By the first iteration we have extracted all the mentioned static and dynamic data to generate the RDF triples. We have created the model and save in as a data set in the Fuseki server. By the final submission, we are going to update the ontology. Because the ontology we created is basic. Even though the ontology is not validated by domain experts, we are going to use and develop the ontology for the better use of RDF graph we generated.

6 Implementation

In order to achieve the goal we have created 2 projects the extractdata project and bicycleSharingStation project. We would explain the reasons and the significance of each project in the below sections.

6.1 Design the model structure

After analyzing the data, we model our scenarios. We identified what are the entities and properties of this specific domain. Since, we could not find a suitable domain ontology for the lyon city guide, we proposed and developed an OWL ontology in Protege.
The below figure depicts the ontology we have created:



6.2 Create RDF model for static Data:

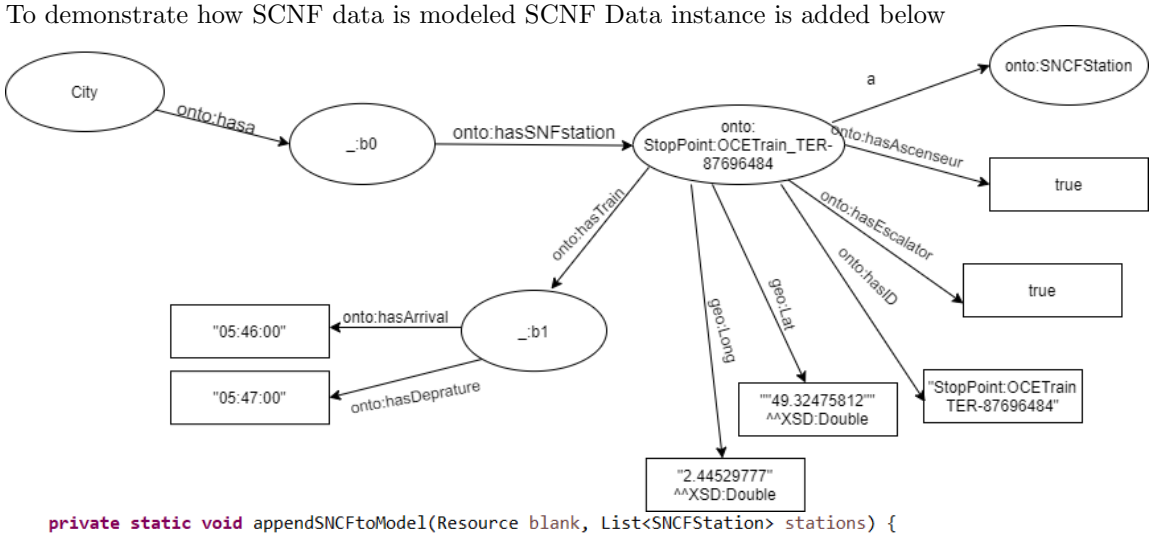
Now, as per our ontology we have created the programmatic models and stored the models in the triple store. Below are the code snippet and detail description of each model which can be seen from the Instances the data:

6.2.1 Hospital model:


```
public static void appendHospitalstoModel(Resource city, List<Hospital> hospitalList) {
    Resource bHospital = model.createResource();
    Property hashospital = model.createProperty(NsPrefix.getOntoNS() + "hashospital");
    Property hasadresse = model.createProperty(NsPrefix.getOntoNS() + "adresse");
    Property hastel_number = model.createProperty(NsPrefix.getOntoNS() + "tel_number");
    Property hascatagorie = model.createProperty(NsPrefix.getOntoNS() + "catagorie");
    Property hasLatitude = model.createProperty(NsPrefix.getGeoNS() + "Lat");
    Property hasLongitude = model.createProperty(NsPrefix.getGeoNS() + "Long");
    Property instanceofhospital = model.createProperty(NsPrefix.getOntoNS() + "instanceofhospital");
    city.addProperty(hashospital, bHospital);
    int i = 0;
    for (Hospital hos : hospitalList) {
        i++;
        Resource Hospital = model.createResource(NsPrefix.getOntoNS() + "hospital" + i);
        Resource hospClass = model.createResource(NsPrefix.getSchemaNS() + "Hospital");
        Hospital.addProperty(RDF.type, hospClass);
        bHospital.addProperty(instanceofhospital, Hospital);
        Hospital.addProperty(hasadresse, hos.getAdresse());
        Hospital.addProperty(hastel_number, String.valueOf(hos.getTel_number()));
        Hospital.addProperty(hasLatitude, String.valueOf(hos.getLat()), XSDDatatype.XSDdouble);
        Hospital.addProperty(hasLongitude, String.valueOf(hos.getLon()), XSDDatatype.XSDdouble);
        Hospital.addProperty(hascatagorie, String.valueOf(hos.getCatagorie()));
    }
}
```

6.2.2 SNCF model:

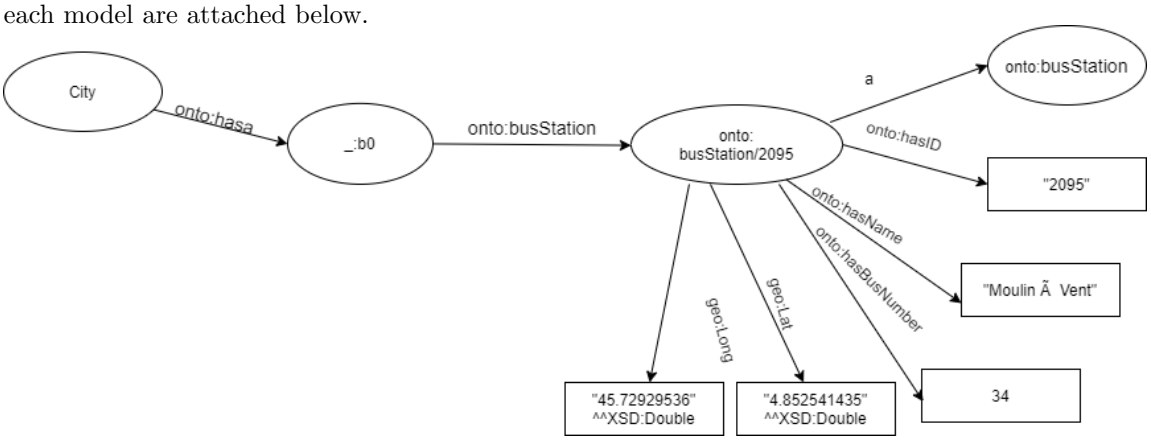
To demonstrate how SCNF data is modeled SCNF Data instance is added below



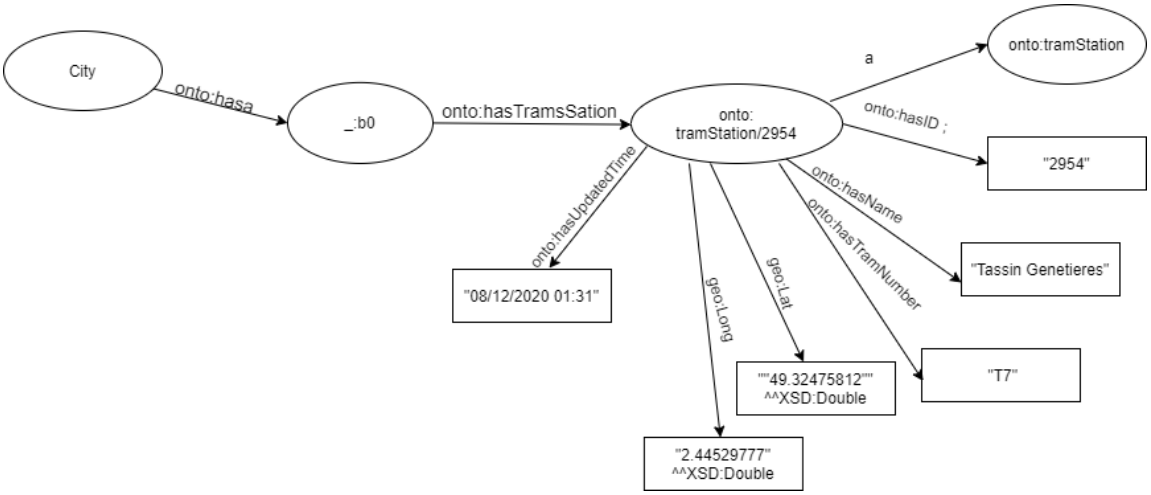

```
private static void appendSNCFtoModel(Resource blank, List<SNCFStation> stations) {

    Property hasSNFstation = model.createProperty(NsPrefix.getOntoNS() + "hasSNFstation");

    Property hasId = model.createProperty(NsPrefix.getOntoNS() + "hasID");
    Property hasName = model.createProperty(NsPrefix.getOntoNS() + "hasName");
    Property hasLatitude = model.createProperty(NsPrefix.getGeoNS() + "Lat");
    Property hasLongtiude = model.createProperty(NsPrefix.getGeoNS() + "Long");
    Property hasEscalator = model.createProperty(NsPrefix.getOntoNS() + "hasEscalator");
    Property hasAscenseur = model.createProperty(NsPrefix.getOntoNS() + "hasAscenseur");
    Property hasTrain = model.createProperty(NsPrefix.getOntoNS() + "hasTrain");
    Property hasArrival = model.createProperty(NsPrefix.getOntoNS() + "hasArrival");
    Property hasDeprature = model.createProperty(NsPrefix.getOntoNS() + "hasDeprature");

    for (SNCFStation station : stations) {
        Resource trainstation = model
                .createResource(NsPrefix.getOntoNS() + "SNCFstation/" + station.getID().replaceAll(" ", "_"));
        Resource train = model.createResource();
        blank.addProperty(hasSNFstation, trainstation);
        Resource trainClass = model.createResource(NsPrefix.getSchemaNS() + "SNCFStation");
        trainstation.addProperty(RDF.type, trainClass);
        trainstation.addProperty(hasId, station.getID());
        trainstation.addProperty(hasName, station.getName());
        trainstation.addProperty(hasLatitude, String.valueOf(station.getLat()), XSDDatatype.XSDdouble);
        trainstation.addProperty(hasLongtiude, String.valueOf(station.getLon()), XSDDatatype.XSDdouble);
        trainstation.addProperty(hasEscalator, String.valueOf(station.isEscalator()), XSDDatatype.XSDboolean);
        trainstation.addProperty(hasAscenseur, String.valueOf(station.isAscenseur()), XSDDatatype.XSDboolean);
        trainstation.addProperty(hasTrain, train);

        train.addProperty(hasArrival, String.valueOf(station.getArrival()));
        train.addProperty(hasDeprature, String.valueOf(station.getDepart()));

    }
}
```

6.2.3 Bus, Tram, Metro model:

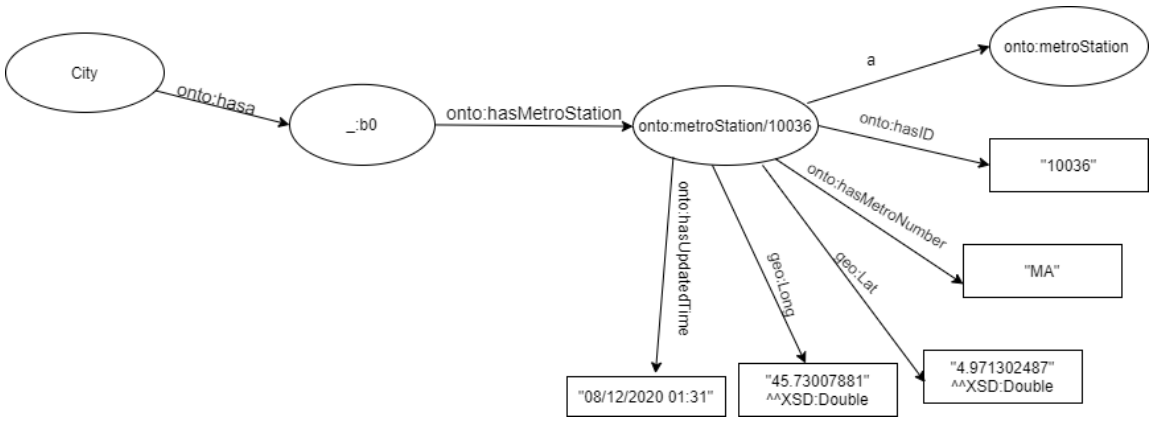
The design that we follow to model Bus,Tram and Metro is same approach to illustrate more,Instances each model are attached below.



Bus data instance



Tram station data instance



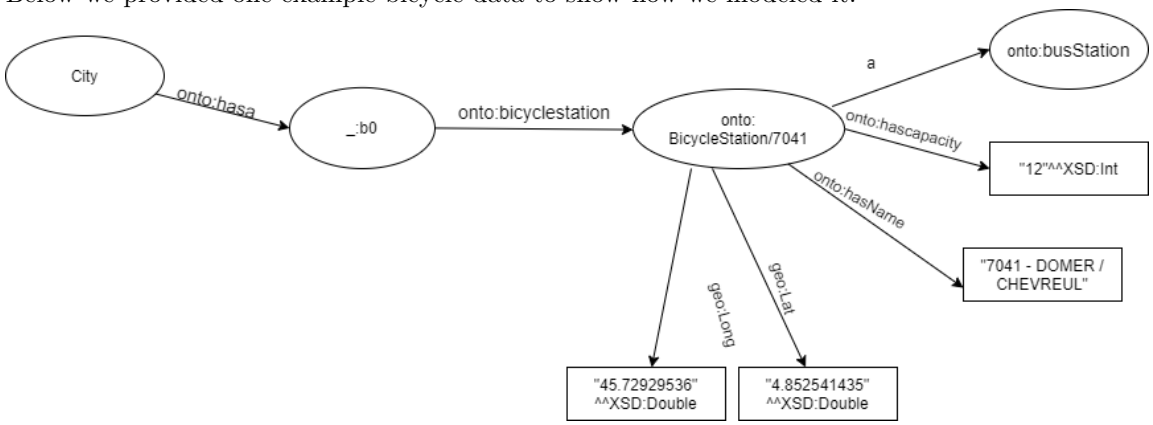
Metro Station data instance

```
private static void appendBTMStoModel(Resource blank, List<BTMStations> stations) {

    Property hasBusStation = model.createProperty(NsPrefix.getOntoNS() + "hasBusStation");
    Property hasTramStation = model.createProperty(NsPrefix.getOntoNS() + "hasTramsSation");
    Property hasMetroStation = model.createProperty(NsPrefix.getOntoNS() + "hasMetroStation");

    Property hasId = model.createProperty(NsPrefix.getOntoNS() + "hasID");
    Property hasName = model.createProperty(NsPrefix.getOntoNS() + "hasName");
    Property hasLatitude = model.createProperty(NsPrefix.getGeoNS() + "Lat");
    Property hasLongtiude = model.createProperty(NsPrefix.getGeoNS() + "Long");
    Property hasBusNumber = model.createProperty(NsPrefix.getOntoNS() + "hasBusNumber");
    Property hasTramNumber = model.createProperty(NsPrefix.getOntoNS() + "hasTramNumber");
    Property hasMetroNumber = model.createProperty(NsPrefix.getOntoNS() + "hasMetroNumber");
    Property hasUpdatedTime = model.createProperty(NsPrefix.getOntoNS() + "hasUpdatedTime");

    for (BTMStations station : stations) {
        Resource busStation = model
                .createResource(NsPrefix.getOntoNS() + "busStation/" + station.getID().replaceAll(" ", "_"));
        Resource tramStation = model
                .createResource(NsPrefix.getOntoNS() + "tramStation/" + station.getID().replaceAll(" ", "_"));
        Resource metroStation = model
                .createResource(NsPrefix.getOntoNS() + "metroStation/" + station.getID().replaceAll(" ", "_"));

        blank.addProperty(hasBusStation, busStation);
        blank.addProperty(hasTramStation, tramStation);
        blank.addProperty(hasMetroStation, metroStation);
```

```
        Resource busClass = model.createResource(NsPrefix.getSchemaNS() + "busStation");

        busStation.addProperty(RDF.type, busClass);
        busStation.addProperty(hasId, station.getID());
        busStation.addProperty(hasName, station.getName());
        busStation.addProperty(hasLatitude, String.valueOf(station.getLat()), XSDDatatype.XSDdouble);
        busStation.addProperty(hasLongtiude, String.valueOf(station.getLon()), XSDDatatype.XSDdouble);
        busStation.addProperty(hasBusNumber, String.valueOf(station.getBusNumber()));
        busStation.addProperty(hasUpdatedTime, String.valueOf(station.getUpdatedtime()));

        Resource tramClass = model.createResource(NsPrefix.getSchemaNS() + "tramStation");
        tramStation.addProperty(RDF.type, tramClass);
        tramStation.addProperty(hasId, station.getID());
        tramStation.addProperty(hasName, station.getName());
        tramStation.addProperty(hasLatitude, String.valueOf(station.getLat()), XSDDatatype.XSDdouble);
        tramStation.addProperty(hasLongtiude, String.valueOf(station.getLon()), XSDDatatype.XSDdouble);
        tramStation.addProperty(hasTramNumber, String.valueOf(station.getTramNumber()));
        tramStation.addProperty(hasUpdatedTime, String.valueOf(station.getUpdatedtime()));

        Resource metroClass = model.createResource(NsPrefix.getSchemaNS() + "metroStation");
        metroStation.addProperty(RDF.type, metroClass);
        metroStation.addProperty(hasId, station.getID());
        metroStation.addProperty(hasName, station.getName());
        metroStation.addProperty(hasLatitude, String.valueOf(station.getLat()), XSDDatatype.XSDdouble);
        metroStation.addProperty(hasLongtiude, String.valueOf(station.getLon()), XSDDatatype.XSDdouble);
        metroStation.addProperty(hasMetroNumber, String.valueOf(station.getMetroNumber()));
        metroStation.addProperty(hasUpdatedTime, String.valueOf(station.getUpdatedtime()));

    }
}
```

6.2.4 Bicycle model:

Below we provided one example bicycle data to show how we modeled it.



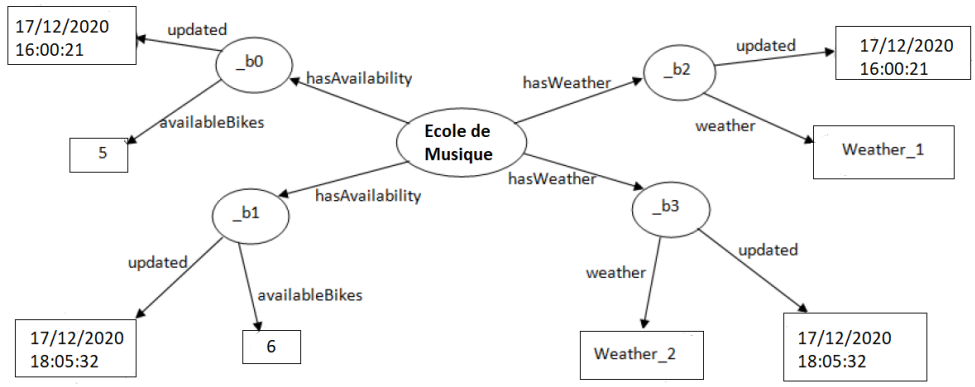
Bicycle data instance

```
public static void appendBicycletoModel(Resource blank, List<BicycleStation> stations) {

    Property hasBicycleStation = model.createProperty(NsPrefix.getOntoNS() + "hasBicycleStation");
    Property hasName = model.createProperty(NsPrefix.getOntoNS() + "hasName");
    Property hascapacity = model.createProperty(NsPrefix.getOntoNS() + "hascapacity");
    int i = 0;
    for (BicycleStation station : stations) {
        i++;
        Resource bicyclestation = model.createResource(NsPrefix.getOntoNS() + "BicycleStation/" + station.getID());
        Resource bicycleClass = model.createResource(NsPrefix.getSchemaNS() + "bicyclestation");

        bicyclestation.addProperty(RDF.type, bicycleClass);
        blank.addProperty(hasBicycleStation, bicyclestation);
        bicyclestation.addProperty(hasName, station.getName(), "En");
        Statement statement_pcapacity = model.createLiteralStatement(bicyclestation, hascapacity,
                station.getCapacity());
        model.add(statement_pcapacity);
        bicyclestation.addLiteral(model.createProperty(NsPrefix.getGeoNS() + "Lat"), station.getLat());
        bicyclestation.addLiteral(model.createProperty(NsPrefix.getGeoNS() + "Long"), station.getLon());

    }
}
```

6.3 Create RDF model for Dynamic Data:

In here we have created model and stored the data in our triple store which includes bicycle data and weather data.

About blank nodes When the availability of bikes is added, we save the history of data. Each time dynamic data extracting process running, we create a blank node for that and add new nodes for the blank node for available bikes and updated date time. We have saved history of weather as well. Each time weather process is running we generated new blank node and give identity for each history data. Below figure depicts the model:



Example of dynamic Bicycle,With updated Availability and Weather-1,Weather-2 represents temperature,humidity,pressure and windspeed

```
String iri = station.getIri();
String nava = station.getNava();
String query = "PREFIX schema: <http://schema.org/> \r\n"
        + "PREFIX geo:   <https://www.w3.org/2003/01/geo/wgs84_pos#> \r\n"
        + "PREFIX rdf:   <http://www.w3.org/2000/01/rdf-schema/> \r\n"
        + "PREFIX onto:  <http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#>\r\n"
        + "INSERT DATA { <" +  iri + ">  a  schema:bicycleStation ;onto:hasAvailability [ \r\n"
        + " a onto:Availability; \r\n"
        + "            onto:updatedDatetime \"" + todayDate + "\" ;\r\n"
        + "        onto:availableBikes \""  + nava + "\" ;\r\n"
        + "].\r\n"
        +"}";


 UpdateRequest update  = UpdateFactory.create(query);
UpdateProcessor qexec = UpdateExecutionFactory.createRemote(update, FUESKI_LOCAL_ENDPOINT);
qexec.execute();
```

```
String query = "PREFIX schema: <http://schema.org/> \r\n"
        + "PREFIX geo:   <https://www.w3.org/2003/01/geo/wgs84_pos#> \r\n"
        + "PREFIX xsd:   <http://www.w3.org/2000/01/rdf-schema/> \r\n"
        + "PREFIX onto:  <http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#>\r\n "
        + "INSERT DATA { <" + Id + "> "
        + "onto:hasWeather [\r\n"
        + " a                                onto:Weather; \r\n"
        + " onto:weatherUpdatedDatetime \"" + todayDate + "\"^^xsd:dateTime;\r\n"
        + " onto:temperature \"" + temperature + "\";"
        + " onto:humadity \"" + humadity + "\";"
        + " onto:windSpeed \"" + windSpeed + "\";"
        + " onto:weatherDescription \"" + weatherDescription + "\";"
        + " onto:pressure \"" + pressure + "\"; ] .}";

UpdateRequest update = UpdateFactory.create(query);
UpdateProcessor qexec = UpdateExecutionFactory.createRemote(update, FUESKI_LOCAL_ENDPOINT_UPDATE);
qexec.execute();
```

6.4 SPARQL Queries to fetch the data from triple store:

In order to display the static data we have used the queries in the backend.The snippet of the queries used are provided in the next page.

6.4.1 Querying hospital data:

As explained earlier for hospital we will query the data that we have planned to displayed in our Interface. So, the data are Category,Latitude,Longitude,Address,Phone number.

```
String query = "PREFIX schema: <http://schema.org/> \r\n"
        + "PREFIX geo:   <https://www.w3.org/2003/01/geo/wgs84_pos#> \r\n"
        + "PREFIX rdf:   <http://www.w3.org/2000/01/rdf-schema/> \r\n"
        + "PREFIX onto:  <http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#>\r\n"
        + "PREFIX ont: <http://purl.org/net/ns/ontology-annot#>\r\n" + "\r\n"
        + "SELECT ?category  ?lat ?lon ?address ?telephone ?hospital\r\n" + "WHERE{\r\n"
        + "  ?hospital onto:catagorie ?category.\r\n" + "  ?hospital geo:Lat ?lat .\r\n"
        + "  ?hospital geo:Long ?lon .\r\n" + "  ?hospital onto:tel_number ?telephone .\r\n"
        + "  ?hospital onto:adresse ?address.\r\n" + "}";

Query qu = QueryFactory.create(query);
QueryExecution q = QueryExecutionFactory.sparqlService(FUESKI_LOCAL_ENDPOINT, qu);
ResultSet results = q.execSelect();
```

6.4.2 Querying SNCF data:

The data that displayed in the Interface are Station name,Latitude,Longitude,Arrival Time,Departure time, hasEscalator, hasElevator.

```
String query = "PREFIX schema: <http://schema.org/> \r\n"
        + "PREFIX geo:   <https://www.w3.org/2003/01/geo/wgs84_pos#> \r\n"
        + "PREFIX rdf:   <http://www.w3.org/2000/01/rdf-schema/> \r\n"
        + "PREFIX onto:  <http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#>\r\n"
        + "PREFIX ont: <http://purl.org/net/ns/ontology-annot#>\r\n" + "\r\n"
        + "SELECT ?y  ?x ?lon ?lat ?station ?at ?dt ?id \r\n" + "WHERE{\r\n" + " \r\n"
        + "  ?sncf onto:hasAscenseur ?y.\r\n" + "  ?sncf onto:hasEscalator ?x.\r\n"
        + "  ?sncf geo:Lat ?lat .\r\n" + "  ?sncf geo:Long ?lon .\r\n" + "  ?sncf onto:hasName ?station .\r\n"
        + "  ?sncf onto:hasID ?id.\r\n" + "  ?sncf onto:hasTrain [].\r\n" + "  [] onto:hasArrival ?at.\r\n"
        + "  [] onto:hasDeprature ?dt.\r\n" + "  \r\n" + "}";

Query qu = QueryFactory.create(query);
QueryExecution q = QueryExecutionFactory.sparqlService(FUESKI_LOCAL_ENDPOINT, qu);
ResultSet results = q.execSelect();
```

6.4.3 Querying Bus data:

The data displayed are bus stop,lat ,long, bus number.

```
if (type.equals("BUSSTATIONS")) {
    query = "PREFIX schema: <http://schema.org/> \r\n"
            + "PREFIX geo:   <https://www.w3.org/2003/01/geo/wgs84_pos#> \r\n"
            + "PREFIX rdf:   <http://www.w3.org/2000/01/rdf-schema/> \r\n"
            + "PREFIX onto:  <http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#>\r\n"
            + "PREFIX ont: <http://purl.org/net/ns/ontology-annot#>\r\n" + "\r\n"
            + "SELECT ?stopName ?id ?lat ?lon ?busNumber \r\n" + "WHERE{\r\n" + "  ?bus onto:hasID ?id .\r\n"
            + "  ?bus onto:hasName ?stopName.\r\n" + "  ?bus geo:Lat ?lat .\r\n" + "  ?bus geo:Long ?lon .\r\n"
            + "  ?bus onto:hasBusNumber ?busNumber .\r\n" + "  \r\n" + "}";
```

6.4.4 Querying Tram data:

The data displayed are tram stop,lat ,long, tram number.

```
else if (type.equals("TRAMSTATIONS")) {
  query = "PREFIX schema: <http://schema.org/> \r\n"
          + "PREFIX geo:   <https://www.w3.org/2003/01/geo/wgs84_pos#> \r\n"
          + "PREFIX rdf:   <http://www.w3.org/2000/01/rdf-schema/> \r\n"
          + "PREFIX onto:  <http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#>\r\n"
          + "PREFIX ont: <http://purl.org/net/ns/ontology-annot#>\r\n" + "\r\n"
          + "SELECT ?stopName ?id ?lat ?lon ?tramNumber \r\n" + "WHERE{\r\n" + "  ?t onto:hasID ?id .\r\n"
          + "  ?t onto:hasName ?stopName.\r\n" + "  ?t geo:Lat ?lat .\r\n" + "  ?t geo:Long ?lon .\r\n"
          + "  ?t onto:hasTramNumber ?tramNumber .\r\n" + "  \r\n" + "}";
```

6.4.5 Querying Metro data:

The data displayed are metro stop,lat ,long, metro number.

```
else if (type.equals("METROSTATIONS")) {
  query = "PREFIX schema: <http://schema.org/> \r\n"
          + "PREFIX geo:   <https://www.w3.org/2003/01/geo/wgs84_pos#> \r\n"
          + "PREFIX rdf:   <http://www.w3.org/2000/01/rdf-schema/> \r\n"
          + "PREFIX onto:  <http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#>\r\n"
          + "PREFIX ont: <http://purl.org/net/ns/ontology-annot#>\r\n" + "\r\n"
          + "SELECT ?stopName ?id ?lat ?lon ?metroNumber \r\n" + "WHERE{\r\n"
          + "  ?metro onto:hasID ?id .\r\n" + "  ?metro onto:hasName ?stopName.\r\n"
          + "  ?metro geo:Lat ?lat .\r\n" + "  ?metro geo:Long ?lon .\r\n"
          + "  ?metro onto:hasMetroNumber ?metroNumber .\r\n" + "  \r\n" + "}";
```

6.4.6 Querying static Bicycle data:

The data queried are station name, Lat, Long, capacity.

```
String query = "PREFIX schema: <http://schema.org/> \r\n"
        + "PREFIX geo:   <https://www.w3.org/2003/01/geo/wgs84_pos#> \r\n"
        + "PREFIX rdf:   <http://www.w3.org/2000/01/rdf-schema/> \r\n"
        + "PREFIX onto:  <http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#>\r\n"
        + "PREFIX ont: <http://purl.org/net/ns/ontology-annot#>\r\n" + "\r\n"
        + "SELECT ?stopname  ?lat ?lon ?capacity \r\n" + "WHERE{\r\n" + " \r\n"
        + "  ?b onto:hasName ?stopname.\r\n" + "  ?b geo:Lat ?lat .\r\n" + "  ?b geo:Long ?lon .\r\n"
        + "  ?b onto:hascapacity ?capacity .\r\n" + "  \r\n" + "}";

Query qu = QueryFactory.create(query);
QueryExecution q = QueryExecutionFactory.sparqlService(FUESKI_LOCAL_ENDPOINT, qu);
ResultSet results = q.execSelect();
```

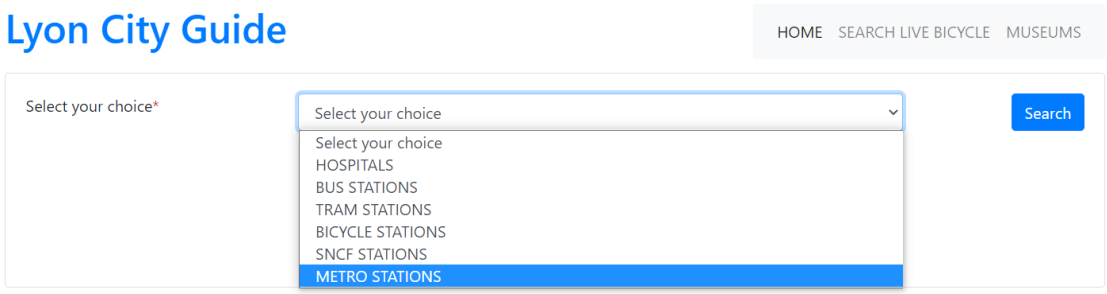
6.5 SPARQL Queries to fetch the data from wiki pedia:

As explained earlier we have also queried wikidata and displayed in our Interface directly.Below is the query snippet:

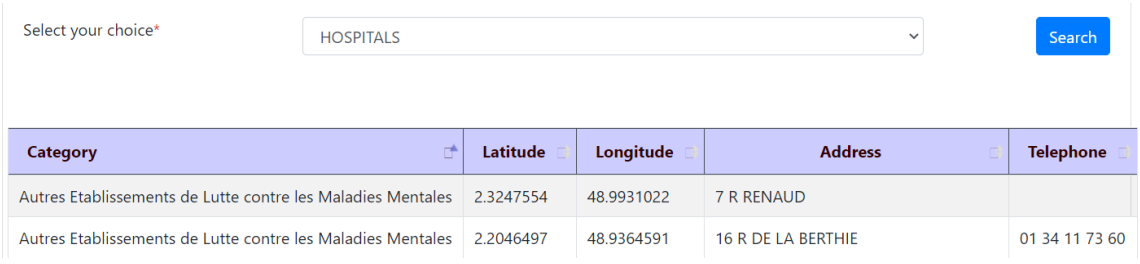
```
List<Museum> list = new ArrayList<Museum>();
String sparqlEndPoint = "https://query.wikidata.org/sparql";

String wikidataPrefixes = "PREFIX bd: <http://www.bigdata.com/rdf#> PREFIX wikibase: <http://wikiba.se/ontology#>  "
        + "PREFIX wdt: <http://www.wikidata.org/prop/direct/>  PREFIX wd: <http://www.wikidata.org/entity/> ";

String queryString = wikidataPrefixes + "SELECT DISTINCT ?museumLabel ?museumDescription ?villeId ?villeIdLabel "
        + "(?villeIdLabel AS ?ville) ?coord ?lat ?lon\r\n"
        + "WHERE\r\n"
        + "{\r\n"
        + "  ?museum wdt:P539 ?museofile.  \r\n"
        + "  ?museum wdt:P131* wd:Q456.\r\n"
        + "  ?museum wdt:P131 ?villeId. \r\n"
        + "  OPTIONAL {?museum wdt:P856 ?link.}   \r\n"
        + "  OPTIONAL {?museum wdt:P625 ?coord .}\r\n"
        + " \r\n"
        + "  SERVICE wikibase:label { bd:serviceParam wikibase:language \"en\". } \r\n"
        + "}\r\n"
        + "ORDER BY  ?villeIdLabel";
QueryExecution qexec = QueryExecutionFactory.sparqlService(sparqlEndPoint, queryString);
```

6.6 User Interface

The website that we have developed has 3 pages in .jsp extention:
Firstly the static data HOME where you can select your choice from the dropdown list:

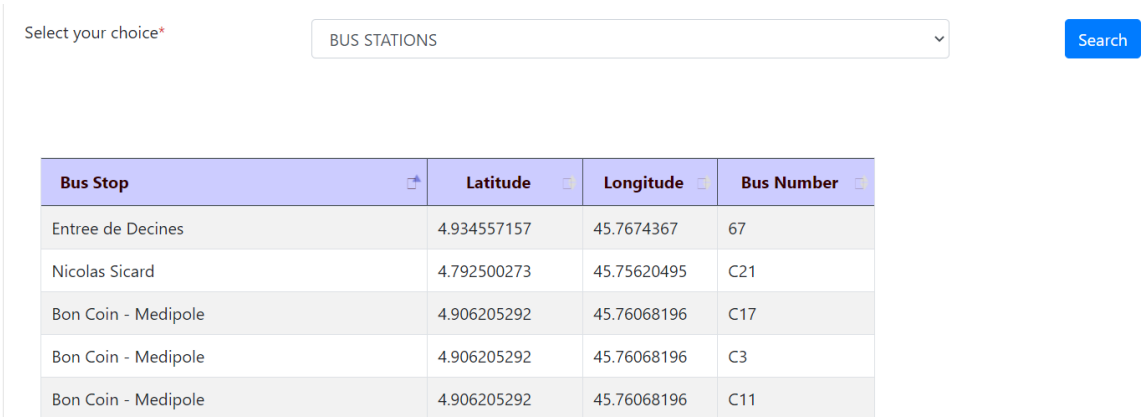


Selected Hospitals option from dropdown

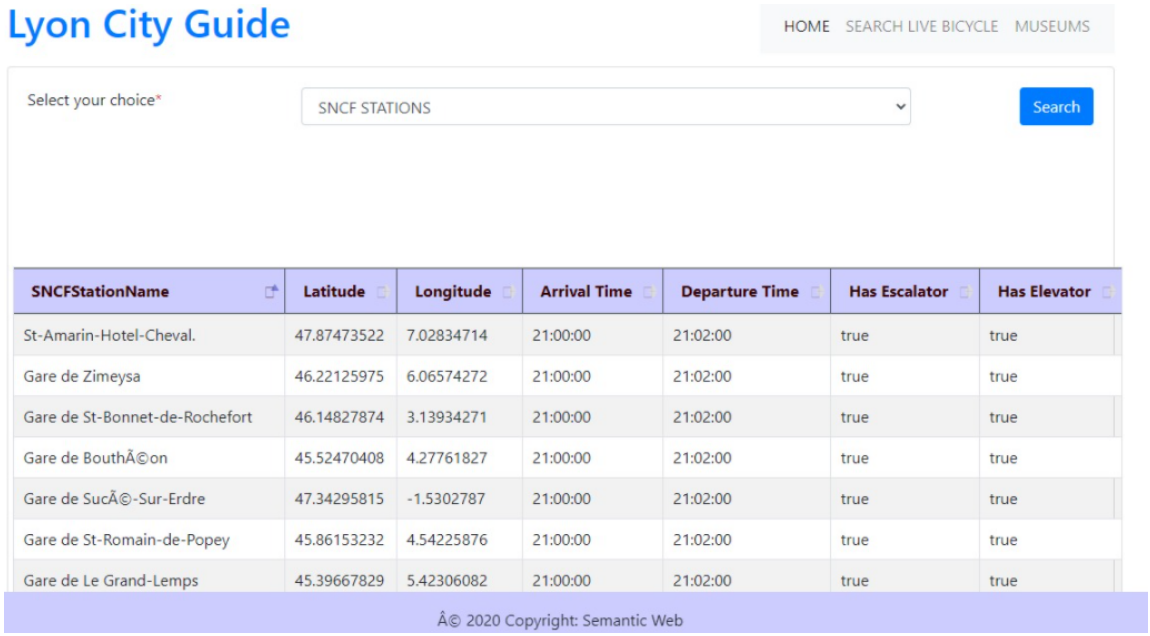


Centres Hospitaliers Régionaux	2.4244092	48.9151135	125 R DE STALINGRAD	01 48 95 55 55
Etablissements d'Enfants à Caractère Sanitaire	2.2625855	48.8932559	41 BD PAUL EMILE VICTOR	01 46 24 97 28
Etablissements d'Enfants à Caractère Sanitaire	2.3154637	48.9021234	49B R KLOCK	01 41 06 98 69
Etablissements d'Enfants à Caractère Sanitaire	2.4344377	48.8677846	4 PL DU GENERAL DE GAULLE	01 49 88 22 55
Hôpitaux Locaux	1.6004891	48.7927308	42 R DE PARIS	01 30 46 18 00
Hôpitaux Locaux	2.334974	49.1038188	2 ALL DE LA FONTAINE AU ROY	01 30 35 51 23
Hôpitaux Locaux	2.606221	48.6941189	17 R PETIT DE BEAUVERGER	01 60 62 62 62
Hôpitaux Locaux	1.9042299	48.8051346	23 R SAINT LOUIS	01 34 91 78 78
Etablissements de Lutte contre l'Alcoolisme	2.4019649	48.7699012	34 BD DE STALINGRAD	01 46 80 11 68
Etablissements de Lutte contre l'Alcoolisme	1.9283484	49.0077886	2 AV DU MARECHAL JOFFRE	01 30 99 96 00
Centres de Lutte contre le Cancer	2.3447836	48.8435418	26 R D'ULM	01 44 32 40 00

Selected Bus station option from dropdown below is a snippet from the lists



Selected SNCF station option from dropdown below is a snippet from the

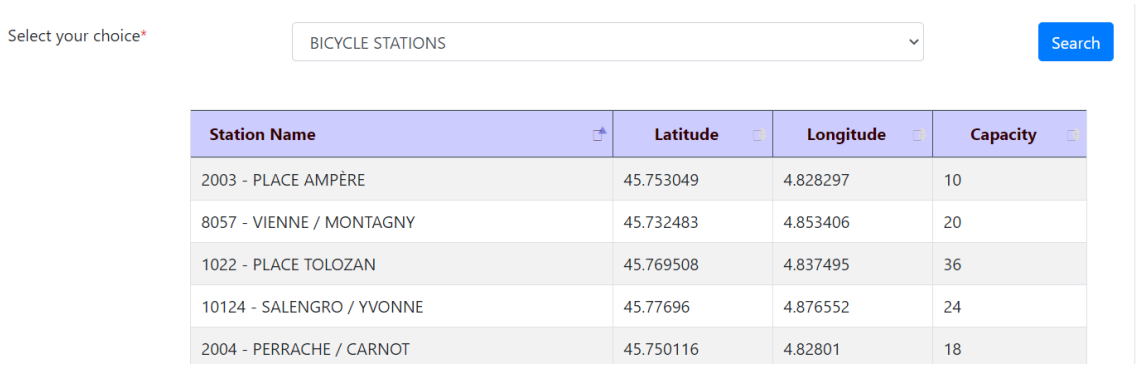


SNCFStationName	Latitude	Longitude	Arrival Time	Departure Time	Has Escalator	Has Elevator
St-Amarin-Hotel-Cheval.	47.87473522	7.02834714	21:00:00	21:02:00	true	true
Gare de Zimeysa	46.22125975	6.06574272	21:00:00	21:02:00	true	true
Gare de St-Bonnet-de-Rochefort	46.14827874	3.13934271	21:00:00	21:02:00	true	true
Gare de BouthÃ©on	45.52470408	4.27761827	21:00:00	21:02:00	true	true
Gare de SucÃ©-Sur-Erdre	47.34295815	-1.5302787	21:00:00	21:02:00	true	true
Gare de St-Romain-de-Popey	45.86153232	4.54225876	21:00:00	21:02:00	true	true
Gare de Le Grand-Lemps	45.39667829	5.42306082	21:00:00	21:02:00	true	true

Selected Tram station option from dropdown below is a snippet from the lists

Tram Stop	Latitude	Longitude	Tram Line
Entree de Decines	4.934557157	45.7674367	T1
Nicolas Sicard	4.792500273	45.75620495	T2
Bon Coin - Medipole	4.906205292	45.76068196	T2
Bon Coin - Medipole	4.906205292	45.76068196	T1

Selected Bicycle station option from dropdown below is a snippet from the lists

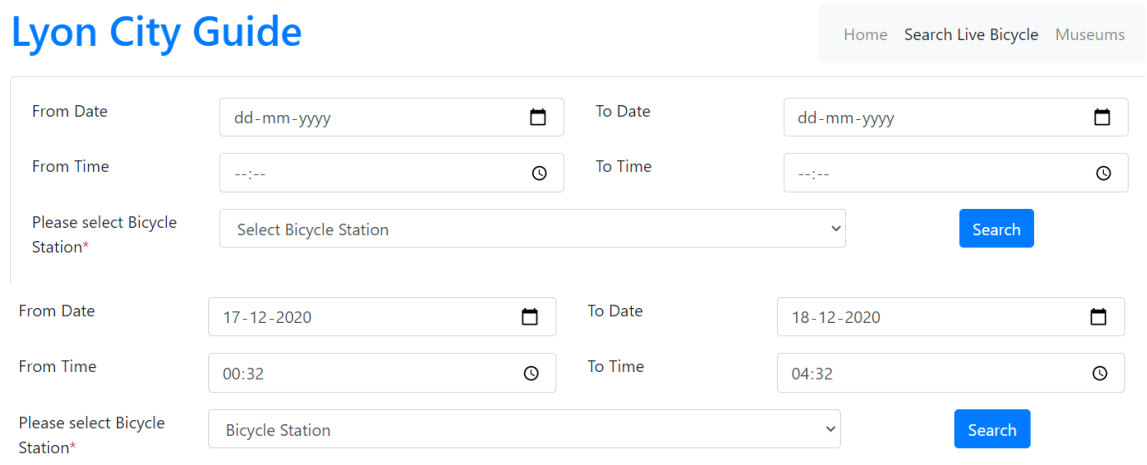


Station Name	Latitude	Longitude	Capacity
2003 - PLACE AMPÈRE	45.753049	4.828297	10
8057 - VIENNE / MONTAGNY	45.732483	4.853406	20
1022 - PLACE TOLOZAN	45.769508	4.837495	36
10124 - SALENGRO / YVONNE	45.77696	4.876552	24
2004 - PERRACHE / CARNOT	45.750116	4.82801	18

Selected Metro stations option from dropdown below is a snippet from the lists

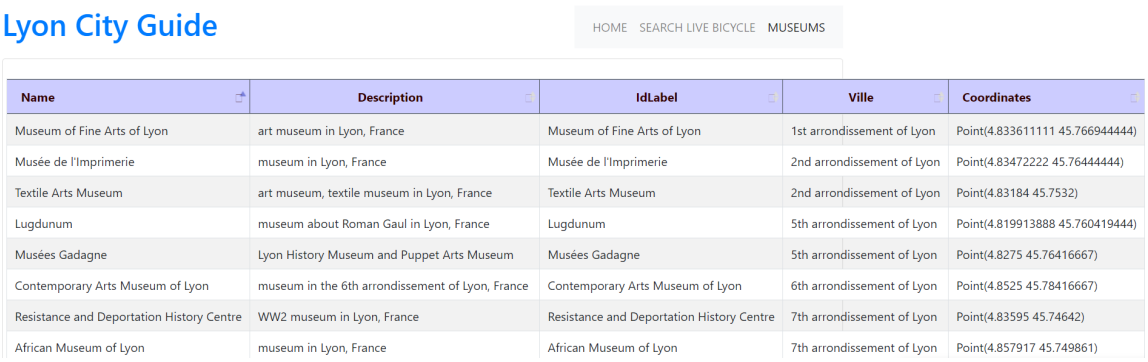
Metro Stop	Latitude	Longitude	Metro Line
Entree de Decines	4.934557157	45.7674367	MA
Nicolas Sicard	4.792500273	45.75620495	MA
Bon Coin - Medipole	4.906205292	45.76068196	MA
Gare de Villeurbanne	4.891167815	45.75622631	MA
Tassin Genetieres	4.77214111	45.7587164	MC
Tassin Genetieres	4.77214111	45.7587164	MA
Berthaudiere	4.969655191	45.77552484	MA
Charton	4.810666671	45.71179575	MA
Aqueducs de Beaunant	4.779963363	45.7243246	MA

Secondly the dynamic data which are being queried from triple store and displayed in UI. Select the next .jsp page Search Live Bicycle



Station Name	Available Bicycle	Temperature (F)	Humidity (%)	WindSpeed (KMH)	Pressure
Point du Jour / Granges	0	277.56	75	3.1	1000
Quai Paul Sédaillan	15	277.55	74	3.1	1000
Ecole de musique	6	277.84	72	3.1	1000
Croix Rousse / Perfetti	15	277.59	73	3.1	1000
Villeurbanne Saint-Jean	4	278.38	72	3.1	1000
Jaurès / Thibaudière	24	277.82	71	3.1	1000
Place Regaud	9	277.81	70	3.1	1000

Thirdly the wikidata page as MUSEUMS below is a small snippet from the list:



Name	Description	IdLabel	Ville	Coordinates
Museum of Fine Arts of Lyon	art museum in Lyon, France	Museum of Fine Arts of Lyon	1st arrondissement of Lyon	Point(4.833611111 45.766944444)
Musée de l'Imprimerie	museum in Lyon, France	Musée de l'Imprimerie	2nd arrondissement of Lyon	Point(4.83472222 45.76444444)
Textile Arts Museum	art museum, textile museum in Lyon, France	Textile Arts Museum	2nd arrondissement of Lyon	Point(4.83184 45.7532)
Lugdunum	museum about Roman Gaul in Lyon, France	Lugdunum	5th arrondissement of Lyon	Point(4.819913888 45.760419444)
Musées Gadagne	Lyon History Museum and Puppet Arts Museum	Musées Gadagne	5th arrondissement of Lyon	Point(4.8275 45.76416667)
Contemporary Arts Museum of Lyon	museum in the 6th arrondissement of Lyon, France	Contemporary Arts Museum of Lyon	6th arrondissement of Lyon	Point(4.8525 45.78416667)
Resistance and Deportation History Centre	WW2 museum in Lyon, France	Resistance and Deportation History Centre	7th arrondissement of Lyon	Point(4.83595 45.74642)
African Museum of Lyon	museum in Lyon, France	African Museum of Lyon	7th arrondissement of Lyon	Point(4.857917 45.749861)

6.7 RDFa used in each web pages:

Initially the prefix geo: and onto: is used in body tag of HTML.Below is a snippet:

```
<body prefix="geo: https://www.w3.org/2003/01/geo/wgs84_pos# onto: http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#">
```

Then, for each of the table which we have defined in the web page. Below is bicycle table example of all the tables.

```
'<tr>'+
    '<td vocab="http://Schema.org/" resource="'+value.name+'" typeof="Station"><span property="dc:title">'+value.name+'</span></td>'+
    '<td resource="'+value.name+'"><span property="geo:lat">'+value.lat+'</span></td>'+
    '<td  resource="'+value.name+'"><span property="geo:lon">'+value.lon+'</span></td>'+
    '<td  resource="'+value.name+'"><span property="onto:capacity">'+value.capacity+'</span></td>'+


'</tr>';
```

Like wise we have the RDFa embedded in all the tables for hospital,bus,tram,SNCF,metro.
The **second web page** has the RDFa for the prefix and data table below is a snippet:

```
'<tr>'+
    '<td vocab="https://opendata.paris.fr/api/records/1.0/search/?dataset=velib-emplacement-des-stations" resource="'+value.stationName+'" typeof="Station"><sp
    '<td prefix="onto: http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#"'+value.stationName+'"><span property="onto:availableBikes">'+value.availab
    '<td prefix="onto: http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#"'+value.stationName+'"><span property="onto:temperature">'+value.temperatur
    '<td prefix="onto: http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#" resource="'+value.stationName+'"><span property="onto:humidity">'+value.hu
    '<td prefix="onto: http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#" resource="'+value.stationName+'"><span property="onto:windspeed">'+value.w
    '<td prefix="onto: http://www.semanticweb.org/emse/ontologies/2020/11/city.owl#" resource="'+value.stationName+'"><span property="onto:pressure">'+value.pr
    '<td>'
'</tr>';
```

The **third web page** has the RDFa in the body tag the prefix is mentioned and for data table below is a snippet:

```
'<tr>'+
    '<td vocab="https://opendata.paris.fr/api/records/1.0/search/?dataset=velib-emplacement-des-stations" resource="'+value.museumLabel+'" typeof="Museum"><span pro
    '<td resource="'+value.museumLabel+'"><span property="museumDescription">'+value.museumDescription+'</span></td>'+
    '<td resource="'+value.museumLabel+'"><span property="museumLabel">'+value.museumLabel+'</span></td>'+
    '<td resource="'+value.museumLabel+'"><span property="ville">'+value.ville+'</span></td>'+
    '<td resource="'+value.museumLabel+'"><span property="coord">'+value.coord+'</span></td>'+

'</tr>';
```

7 Conclusion

In this project we have used all the semantic web technologies we have came across during the course, In order to prove our work we have also developed an Interface which can be used as demo. We have also prepared a small demo video which explains our work in very short span. Of course in this project we have room to add more and more data that we want to add. This project is just an instance of big ontology which will includes all the domains. By, doing this project we have gained a vast knowledge about RDF data model , SPARQL queries and importance of structured data.

8 References

- https://www.w3.org/TR/rdf-sparql-query/
-https://spring.io/guides/gs/spring-boot/
- https://jena.apache.org/tutorials/rdf-api.html
-https://spring.io/guides/gs/rest-service/

9 Appendix

OWL model implementation in Protege.
The owl file can be found in our project folder extractData. The name of the file is **creatmodel-Full.ttl**