INTRODUCTION

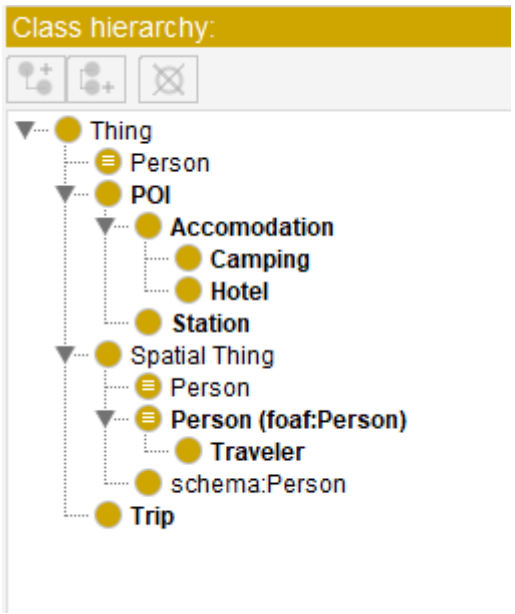
Notre étude se porte sur les gares TGV ainsi que les hôtels/camping. Durant ce projet nous avons pu manipuler des fichiers RDF, OWL, CSV ainsi que des logiciels tels que Protégé, Google Colab, Eclipse, Django et d'autres.

Part I: Modeling the ontology

Dans cette partie nous avons modélisé notre ontologie. Nous avons importé foaf pour l'utiliser dans notre ontologie pour la classe Person.

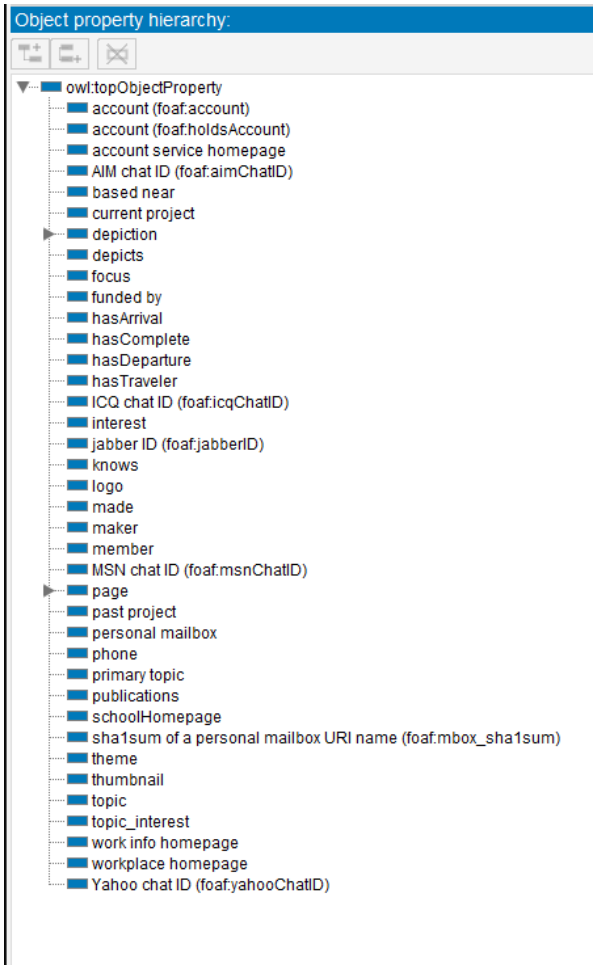
Nous avons donc 3 classe majeure : POI, Trip et Person. Dans chacune de ces classes il y a des sous classes Accommodation, Camping, Hôtel, Station et Traveler. Comme vous pouvez le voir ci-dessous.

a. Définition des classes

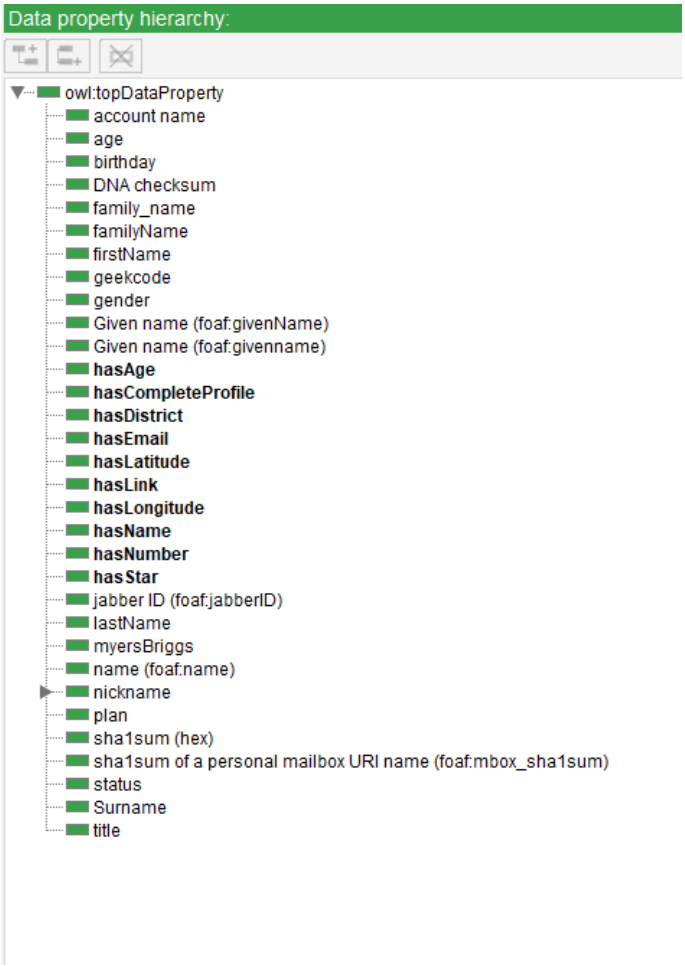


b. Définition de datatype et object properties

Nous avons défini ci-dessous des propriétés d'objet. Donc par exemple hasArrival qui a un domain de la classe POI.



Nous avons aussi défini des data type property pour les classes. En effet nous avons par exemple hasLogitude qui a pour domain POI et range un type float.



c. Définition des restrictions

Ici nous avons défini des restrictions comme avoir exactement une destination d'arrivé. Ces restrictions servent à avoir une ontologie qui a du sens.

Part II: Populating the ontology

Afin de crée une population nous avons réalisé des scripts python dans l'objectif de convertir les différents csv en RDF. Voici une partie du script python utilisé pour peupler notre base de données. Avant de convertir les csv en RDF nous avons réalisé un travail de nettoyage de données sur notre dataset. En effet certaines valeurs dans le dataset n'était pas renseigner, de ce fait nous avons enlevé les gares ou par exemple il n'y avait pas de coordonnée de latitude ou de longitude.

```
[ ] gares
```

	index	Nom_Gare	INSEE_REG	NOM_REG	INSEE_DEP	NOM_DEP	Latitude	Longitude
0	1.0	Aix-en-Provence TGV	84	Auvergne-Rhône-Alpes	1	Ain	46.097	4.847
1	2.0	Bellegarde	84	Auvergne-Rhône-Alpes	1	Ain	46.114	5.824
2	3.0	Bourg-en-Bresse	84	Auvergne-Rhône-Alpes	1	Ain	46.196	5.275
3	4.0	Cluses	84	Auvergne-Rhône-Alpes	1	Ain	46.171	5.577
4	5.0	Nurieux	84	Auvergne-Rhône-Alpes	1	Ain	46.181	5.521
...	...	...	...	...	...	...	...	...
179	179.0	Saint-Dié-des-Vosges	44	Grand Est	88	Vosges	48.293	6.962
180	180.0	Saint-Michel - Valloire	44	Grand Est	88	Vosges	48.157	6.134
181	181.0	Belfort - Montbéliard TGV	27	Bourgogne-Franche-Comté	90	Territoire de Belfort	47.637	6.858
182	182.0	Massy - Palaiseau	11	Île-de-France	91	Essonne	48.731	2.243
183	183.0	Massy TGV	11	Île-de-France	91	Essonne	48.736	2.291

184 rows × 8 columns

```
pro=""

for i in gares.iterrows():
    nom=i[1]["Nom_Gare"]
    lon=i[1]["Longitude"]
    lat=i[1]["Latitude"]
    dep=i[1]["NOM_DEP"]
    owl=f'<owl:NamedIndividual rdf:about="http://www.owl-ontologies.com/unnamed.owl#{str(nom)}">\n<rdf:type rdf:resource="http://www.owl-ontologies.cor
    pro+=owl
```

Part III: Querying the ontology

2. Write SPARQL queries to response to the following:
    a. List the instances of the geolocated POI

SELECT * WHERE {

{

?poi rdf:type ns:Station} union {?poi rdf:type ns:Camping} union {?poi rdf:type ns:Hotel}.

?poi ns:hasLatitude ?lat.

?poi ns:hasLongitude ?long.

?poi ns:hasName ?name.

?poi ns:hasDistrict ?district.

}

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

SELECT * WHERE {,

?poi rdf:type ns:Station.

```
?poi ns:hasName ?name.
?poi ns:hasDistrict ?city.
}
```

c. List the name of trips that have Paris (or any other chosen city) as destination.

```
SELECT * WHERE {
 ?t rdf:type ns:Trip.
 ?t ns:hasDeparture ?dep.
 ?t ns:hasArrival ?ar.
 ?ar ns:hasName ?arName.
 FILTER(?arName='Paris-Gare-de-Lyon')

}
```

d. List the name of travellers older than 22 years.

```
SELECT * WHERE {
 ?poi rdf:type ns:Traveler.
 ?poi ns:hasName ?Name.
 ?poi ns:hasAge ?Age.
FILTER(?Age>22)
}
```

3. Propose 5 SPARQL queries:
   a. A query that contains at least 2 Optional Graph Patterns

List name of travellers with optional properties

```
SELECT * WHERE {
```

```
?tr rdf:type ns:Traveler.
?tr ns:hasName ?name.
OPTIONAL { ?tr ns:hasAge ?age. }
OPTIONAL {?tr ns:hasEmail ?email.}
}
```

b. A query that contains at least 2 alternatives and conjunctions

Get traveler from join to trip object

```
SELECT * WHERE {
?t rdf:type ns:Trip.
?t ns:hasTraveler ?tra.
?tra ns:hasName ?name.
?tra ns:hasAge ?age.

}
```

c. A query that contains a CONSTRUCT query form

Construct query : Construct RDF triple of all station that contains District

```
CONSTRUCT {
?station rdf:type ns:Station.
}
WHERE {?station ns:hasDistrict ?capital.}
```

d. A query that contains an ASK query form

ASK query : Search if Berlin is a district of station. It will return true or false

```
ASK {SELECT * WHERE { ?tr rdf:type ns:Station.
```

?tr ns:hasDistrict " Berlin".

}}

e. A query that contains a DESCRIBE query for

DESCRIBE query : Describe content of Station as RDF-triples

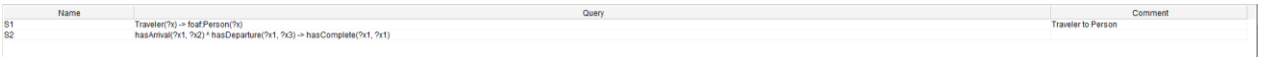
DESCRIBE ?tr

WHERE { ?tr rdf:type ns:Station.

}

4. Define some SWRL rules

Nous avons créé des règles SWRL, dans notre contexte nous avons utilisé des règles pour dire qu'un Traveler devenait aussi une personne mais aussi une règle qui permet de dire qu'un Trip est complet si on a défini un départ et une arrivée



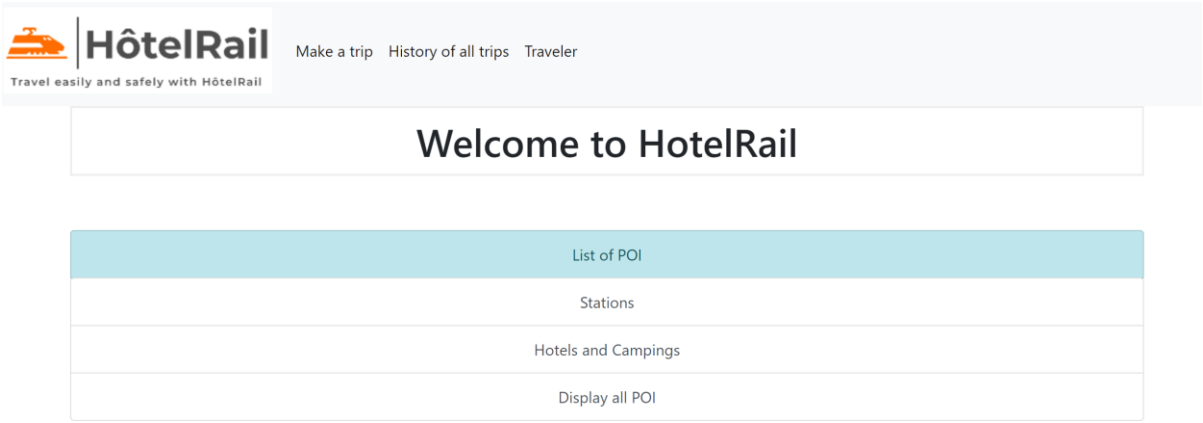
Name	Query	Comment
S1	Traveler(?x) -> foaf:Person(?x)	Traveler to Person
S2	hasArrival(?x1, ?x2) ^ hasDeparture(?x1, ?x3) -> hasComplete(?x1, ?x1)	

Traveler(?x) -> Person(?x),

hasArrival(?x1, ?x2) ^ hasDeparture(?x1, ?x3) -> hasComplete(?x1, ?x1)

Part IV: Manipulating the ontology using Python

Nous avons décidé de créer l'application sous python en utilisant le framework Flask. Python était pour nous la meilleure option car c'est un langage de programmation très utilisé par les 3 membres de l'équipe. De plus il nous permettait de lier facilement les requêtes SPARQL à notre plateforme en ligne.



L'application est disponible sur le GitHub, les explications d'installations sont disponibles sur le README.

L'application nous permet de visualiser les différents POI sur une carte, pour cela nous avons utilisé la librairie Folium.

Les POI ont été stocké dans un fichier rdf, la librairie rdflib nous a permis d'effectuer des requêtes SPARQL et de rechercher les informations nécessaires dans le fichier rdf.