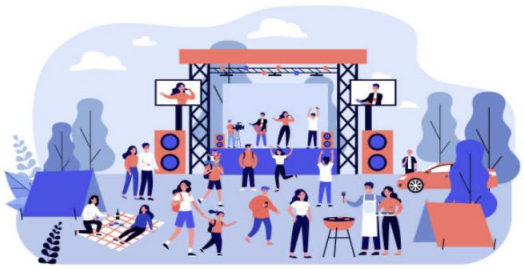
Rapport Projet Ingénierie des Connaissances



Festival City

Un service qui permet d'accéder aux informations des festivals de France et des villes dans lesquels ils se déroulent.

MERIENNE Alexis
SHIM Chaeyeon

25/01/2023

Nom des enseignants :

Catherine FARON
Oscar RODRIGEZ ROCHA
Franck MICHEL

Sommaire

Abstract

Ce projet demande de développer une application dans laquelle on intègre les techniques d'extractions et de représentations de données structurées. Les principales tâches du projet sont :

- La création d'une ontologie OWL et SKOS spécifique pour les données du projet.
- La population de l'ontologie avec des données non structurées trouvées sur le web et des données structurées.
- L'alignement et le liage des vocabulaires et des données avec des ressources sur le web de données liées.
- La caractérisation de la structure du graphe de connaissances avec des contraintes SHACL.
- L'exploitation du graphe de connaissances à travers des requêtes SPARQL pour les principales fonctionnalités de l'application.
- La mise en place d'une interface pour communiquer avec les utilisateurs cibles.

Le sujet que nous avons choisi est celui-ci :

Un système de recommandation de festival se basant sur la localité de ceux-ci et permettant d'obtenir des informations sur les villes dans lesquels se déroulent les festivals.

1. Les sources de données

a) API

Nous avons extrait les informations relatives aux festivals grâce à une API présente sur le site du ministère de la culture est répertoriant tous les festivals se déroulant en France.

Les attributs que nous avons gardé dans le cadre de notre projet sont :

- *Recordid*, un identifiant unique pour chaque festival
- *commune_principale_de_deroulement*
- *nom_du_festival*
- *code_postal_de_la_commune_principale_de_deroulement*
- *region_principale_de_deroulement*

L'API est diponible par le l'URI : https://data.culture.gouv.fr/api/records/1.0/search/?dataset=festivals-global-festivals-_-pl&q=&facet=region_principale_de_deroulement&facet=departement_principal_de_deroulement&facet=commune_principale_de_deroulement&facet=discipline_dominante

b) CSV

Notre CSV décrit les effectifs d'agents municipaux par commune de France. Le document est disponible sur la platforme de données en libre-service du gouvernement. Les attributs sont :

- *nom_commune*
- *nombre_habitant*
- *nombre_police*
- *nombre_asvp*
- *nombre_garde_champetre*

c) Textes

Les textes sont des articles de journaux décrivant les festivals. Ils donnent des informations sur les artistes qui s'y sont produits.

2. Extraction de données

Structure du code

L'extraction des sources de données se fait dans le dossier "engine". Ce dossier est composé de 3 sous-dossiers :

- csv2rdf-lifting
- sparql2rdflifting
- Txt2rdflifting

Chacun de ces dossiers présente un fichier python représentant l'objet d'extraction de données. Ces objets sont appelés dans le ficher *engine.py*.

a) API

Nous utilisons le sparql-microservice pour extraire les données d'API sous forme de graphe de connaissance. Notre requête CONSTRUCT est la suivante :

```
@prefix schema: <http://schema.org/>
@prefix : <http://projetwebsemantique.org/festival/schema#>

CONSTRUCT {

  ?festivaluri a :Festival;
    :located ?communeuri;
    :name ?nomFestival ;
    :hasRegion ?region .
  ?communeuri a :City ;
    :name ?commune .

} WHERE {

  ?festival
  api:recordid ?id;
  api:fields [
    api:commune_principale_de_deroulement ?commune;
```

```
    api:nom_du_festival ?nomFestival;
    api:code_postal_de_la_commune_principale_de_deroulement ?id_commune;
    api:region_principale_de_deroulement ?region
  ] .

  bind (IRI(concat("http://projetwebsemantique.org/festival/data#",
?id)) AS ?festivaluri)
  bind (IRI(concat("http://projetwebsemantique.org/festival/data#",
?id_commune)) AS ?communeuri)
}
```

On populе notre graphe RDF avec deux types de sujets. Le premier est un festival, de type :Festival. Son URI est construite à partir de l'identifiant du festival renseigné par l'API. Le second est une ville, de type :City. Son URI est construite à partir du code postal de la ville.

b) CSV

On construit le graphe de connaissance à partir d'un fichier csv organisé comme ci-dessous:

Nom_commune	Nombre_habitant	Nombre_police	Nombre_asvp	Nombre_garde_champetre
Ambérieu en Bugey	14586	8	0	0
Ambérieux en Dombes	1792	0	0	1
Ambronay	2915	0	1	0
Anglefort	1130	0	1	0
Arbent	3367	3	0	0
Attignat	3300	1	0	0
Bage dommartin	4171	1	0	0
Balan	2615	1	0	0
Belley	9520	5	2	0

Les informations qui nous intéressent sont le nom de la commune, le nombre d'habitant, l'effectif de police et éventuellement l'effectif ASVP.

Malheureusement, ce dataset nous donne pas le code postal de la commune, nous ne pouvons donc pas enregistrer la commune de la même manière que pour le lifting par l'API. Nous avons fait le choix de construire un autre graphe de connaissance avec les ressources décrivant des villes étant enregistré avec un URI contenant leur nom. Nous savons que ce choix n'est pas le meilleure et qu'une façon optimale aurait été de construire un graphe de connaissance décrivant une même ressource avec une URI unique.

A partir de chaque ligne, on construit une ressource :City, qui est décrit avec les propriétés :CityHasName, :nbrHabitant, nbrPolice, nbrASVP.

c) Texte

Pour extraire des informations à partir des articles de journaux décrivant les festivals, nous utilisons l'outil dbspotlight, qui à partir d'un texte, reconnait des ressources présentent dans son graphe de connaissance. On extrait donc des URI décrivant des groupes de musique ou des artistes solo à partir de ces articles de journaux.

On utilise une requête UNION pour obtenir les groupes ET les artistes solos qui sont présent dans le graphe DBpedia.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
          SELECT *
          WHERE {
          {
              <"""+uri+"""> a ?c .
              <"""+uri+"""> dbp:name ?name .
              FILTER (?c = dbo:Band)
          }
          UNION
          {
              <"""+uri+"""> a ?c .
              <"""+uri+"""> dbp:name ?name .
              FILTER (?c = dbo:Artist)
          }
          }
```

d) Alignement des données

Les ressources :Artist et :Group dbpedia sont ensuite alignées avec les données du graphe des festival. En d'autres termes, pour chaque article de journal associé à un festival, on cherche la ressource correspondant au festival dans notre graphe de connaissance et on ajoute, par le biais d'une requête INSERT, les artistes ou groupe qui s'y produisent.

Par la même occasion, on construit la ressource (:Artist || :MusicGroup), dans notre graphe de connaissance.

```
PREFIX : <http://projetwebsemantique.org/festival/schema#>
          INSERT DATA {
              <"""+uri+"""> :hasArtist <"""+artist_uri+"""> .
              <"""+artist_uri+"""> a :"""+artist_type+""" .
              }
```

Avec :

- *uri*, l'URI du festival dans notre graphe de connaissance
- *artist-uri*, l'URI dpbedia décrivant le groupe ou l'artiste solo
- *artist_type*, le type de la ressource, soit :Artsist, soit :MusicGroup

3.Ontologie

La description d'ontologie OWL des classes de notre projet se trouve ici : *datastorage/festival_owl.ttl*

Classes OWL

```
@prefix ftl: <http://projetwebsemantique.org/festival/schema#> .
```

ftl:Festival	Restriction sur la propriété located. Toutes les ressources sont de class :City
ftl:City	Classe représentant les villes
ftl:MusicEntity	Union des classes MusicGroup et Artist
ftl:MusicGroup	Classe représentant les groupes de musique
ftl:Artist	Classe représentant les artistes solos

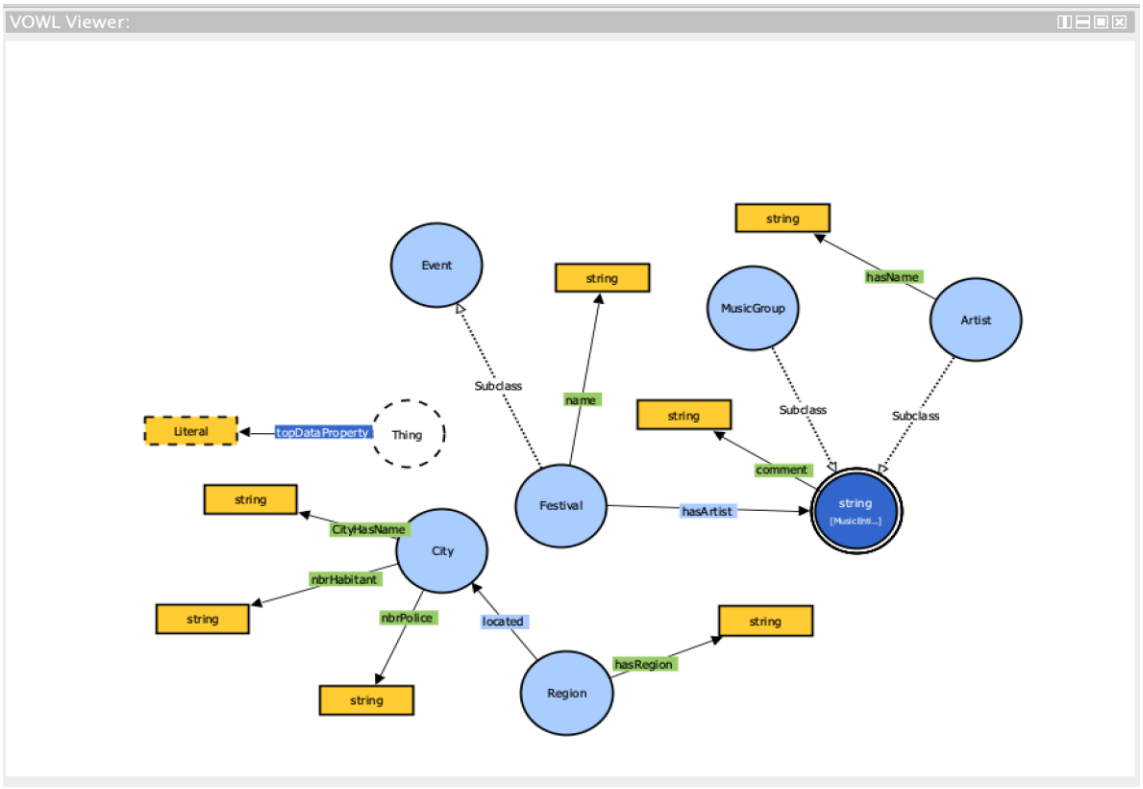
Propriétés OWL

ftl:name a owl:DatatypeProperty ;	rdfs:comment "name of the festival" ; rdfs:domain ftl:Festival,ftl:City ; rdfs:range xsd:string .
ftl:located a owl:ObjectProperty ;	rdfs:comment "the festival is " ; rdfs:domain ftl:Festival ; rdfs:range ftl:City .
ftl:CityHasName a owl:DatatypeProperty ;	rdfs:comment "name of the city" ; rdfs:domain ftl:City ; rdfs:range xsd:string .
ftl:hasName a owl:DatatypeProperty ;	rdfs:comment "name of music entity" ; rdfs:domain ftl:MusicGroup,ftl:Artist ; rdfs:range xsd:string .
ftl:hasRegion a owl:DatatypeProperty ;	rdfs:comment "region where the festival is" ;

	owl:domain ftl:Festival ; owl:range xsd:string.
ftl:hasArtist rdf:type owl:ObjectProperty;	rdfs:comment "artists present during festival"; rdfs:domain ftl:Festival ; rdfs:range ftl:MusicEntity.
ftl:comment a owl:DatatypeProperty ;	rdfs:comment "comment" ; rdfs:domain ftl:MusicEntity ; rdfs:range xsd:string .
ftl:nbrHabitant a owl:DatatypeProperty ;	rdfs:comment "nombre d'habitants" ; rdfs:domain ftl:City ; rdfs:range xsd:string .
ftl:nbrPolice a owl:DatatypeProperty ;	rdfs:comment "effectif de la police" ; rdfs:domain ftl:City ; rdfs:range xsd:string .

Visualisation de l'ontologie

Nous avons défini plusieurs classes telles que MusicGroup, Artist, City, Festival et des propriétés telles que hasArtist, hasRegion, located, nbrHabitant, nbrPolice, name, hasName. L'ontologie définit ce qui est lié aux festivals de musique et aux entités qui y participent. Pour visualiser notre ontologie, Protégé et VOWL (Visual Notation for OWL Ontologies) nous aident à interpréter ces connaissances. Ainsi nous avons installé le plugin Protégé VOWL. : *http://vowl.visualdataweb.org/protegevowl.html*

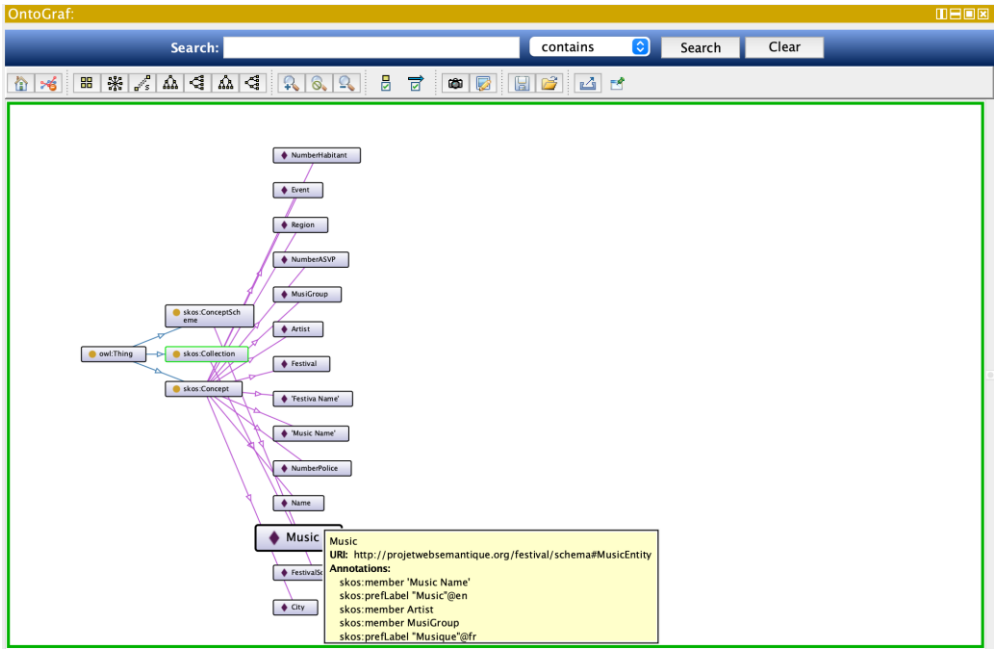


Graphe de l'Ontologie avec VOWL

Concepts SKOS

Pour définir un ensemble de concepts et leurs relations en utilisant le vocabulaire SKOS dans notre ontologie, la description de skos des concepts de notre projet se trouve ici : *datastorage/festival_skos.ttl*

ftl:FestivalScheme rdf:type skos:ConceptScheme.	Definition du schéma concept "FestivalScheme" qui est une collection de concepts et leur relations .
ftl:Event rdf:type skos:Concept;	sskos:topConceptOf ftl:Festival; skos:narrower ftl:Festival.
ftl:Festival rdf:type skos:Concept;	skos:broader ftl:Event; skos:topConceptOf ftl:Region , ftl:City; skos:inScheme ftl:FestivalScheme.
ftl:MusicGroup rdf:type skos:Concept;	skos:hasTopConcept ftl:MusicEntity.
ftl:MusicEntity rdf:type skos:Collection;	skos:member ftl:MusicGroup , ftl:Artist, ftl:MusicName.
ftl:Artist rdf:type skos:Concept;	skos:hasTopConcept ftl:MusicEntity.
ftl:MusicName rdf:type skos:Concept;	skos:hasTopConcept ftl:MusicEntity.
ftl:City rdf:type skos:Concept;	skos:broader ftl:Region; skos:narrower ftl:Name.
ftl:Region rdf:type skos:Concept;	skos:narrower ftl:City.
ftl:Name rdf:type skos:Concept;	skos:broader ftl:Region, ftl:City.
ftl:FestivalName rdf:type skos:Concept;	skos:hasTopConcept ftl:Festival.
ftl:nbrPolice rdf:type skos:Concept;	skos:hasTopConcept ftl:City.
ftl:nbrHabitant rdf:type skos:Concept;	skos:hasTopConcept ftl:City.
ftl:nbrASVP rdf:type skos:Concept;	skos:hasTopConcept ftl:City.



Visualisation de structure des concepts et de la collection , du schéma SKOS

4. SHACL

Pour vérifier que le graphe est conforme aux contraintes shacl édictées, nous utilisons la librairie python pyshacl, qui à partir d'un fichier turtle décrivant des données RDF et un fichier turtle décrivant les contraintes, permet de valider la conformité des données.

Nous testons cette conformité grâce au script python *engine/shacl_validator.py*

Les contraintes sont rédigées dans le fichier turtle *datastorage/festival_shacl.ttl*

5. Requêtes SPARQL intéressantes

Nous avons établi trois requêtes SPARQL complexes qui permet de visualiser les capacités des représentations de nos données

1. Pour chaque festival, donner le nombre de festival qui s'y déroule. Classer le résultat par ordre décroissant.

```
prefix ftl: <http://projetwebsemantique.org/festival/schema#> .
prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

SELECT (COUNT(?x) as ?nbrfest) ?region
where {
    ?x a ftl:Festival
    ?x ftl:hasRegion ?region
    }
GROUP BY ?region
ORDER BY desc(?nbrfest)
```

2. Pour chaque ville, donner le nombre d'artistes solo qui s'y produit. Classer le résultat par ordre décroissant.

```
prefix ftl: <http://projetwebsemantique.org/festival/schema#> .
prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

SELECT (COUNT(?artists) AS ?nbrArtist)  ?nameville
where {
    ?x a ftl:Festival
    ?x ftl:located ?ville
    ?ville ftl:name ?nameville
    ?x ftl:hasArtist ?artists
    MINUS{?artists a ftl:MusicGroup}
    }
GROUP BY ?ville
ORDER BY DESC(?nbrArtist)
```

3. Pour chaque région, donner le nombre de moyen d'agent de police par commune.

```
prefix ftl: <http://projetwebsemantique.org/festival/schema#> .

select (AVG(?police) AS ?avgPolice) ?region
where {

    ?fest a ftl:Festival
    ?fest ftl:located ?city
    ?fest ftl:hasRegion ?region
    ?city ftl:CityHasName ?name_1
    ?city_mun a ftl:City
    ?city_mun ftl:CityHasName ?name_1
    ?city_mun ftl:nbrPolice ?police

}GROUP BY ?region
```

6. L'interface Web

L'interface web permet de réaliser deux tâches :

- Obtenir les festivals se déroulant dans une ville, les :MusicEntity se produisant dans ces festivals et la description de ceux-ci
- Obtenir les festivals se déroulant dans une région, les :MusicEntity se produisant dans ces festivals et la description de ceux-ci

L'application est réalisée avec le framework python flask. Nous avons fait le choix de définir deux classes permettant de requêter nos graphes de connaissance :

- Copservive.py, requête relative aux informations sur les villes. (population et effectif de police)

```
PREFIX s: <http://projetwebsemantique.org/festival/schema#>
PREFIX schema: <http://schema.org/>
SELECT *
        WHERE {
            ?x a s:City .
            ?x
<http://projetwebsemantique.org/festival/schema#CityHasName> ?y .
            ?x
<http://projetwebsemantique.org/festival/schema#CityHasName>
'"""+city_name+"""' .

            ?x
<http://projetwebsemantique.org/festival/schema#nbrPolice> ?nbrpolice .
            ?x  <http://projetwebsemantique.org/festival/schema#nbrASVP>
?nbrasvp .
            ?x
<http://projetwebsemantique.org/festival/schema#:nbrHabitant>
?population .
        }
```

- Festservice.py, requête relative aux informations des festivals.

```
PREFIX s: <http://projetwebsemantique.org/festival/schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT *
        WHERE {
            ?x <http://projetwebsemantique.org/festival/schema#hasRegion
'"""+region+"""'^^xsd:string.
            ?x <http://projetwebsemantique.org/festival/schema#name>
?name .
            OPTIONAL {?x
<http://projetwebsemantique.org/festival/schema#hasArtist> ?artists .}
        }
```

Ces deux classes sont appelées dans app.py.

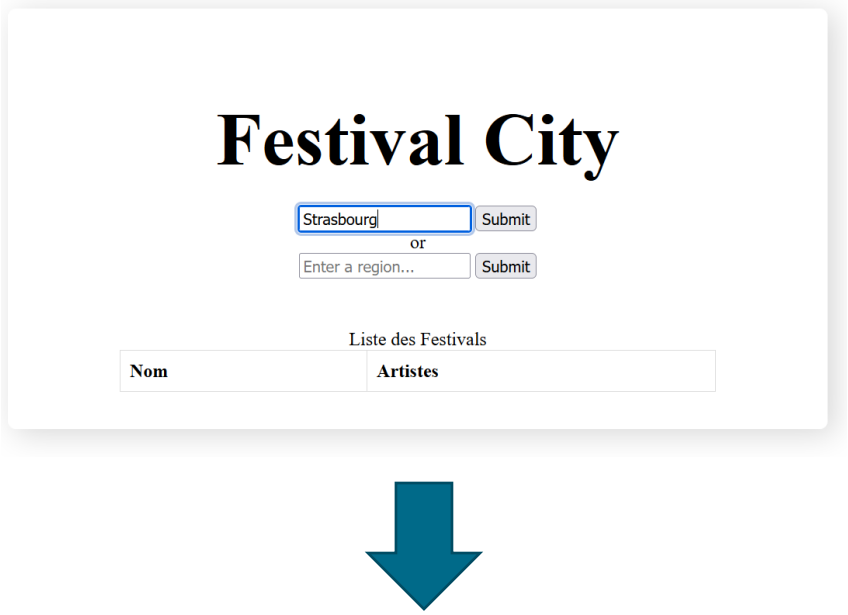
Pour obtenir les informations relatives aux artistes et aux groupes, nous avons établi une *federated query* sur le endpoint de dbpedia, qui permet d'avoir l'objet de la propriété *dbp:comment* à partir de l'URI décrivant une ressource dbpedia.

```
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
                    SELECT *
                    WHERE {
                    SERVICE <http://dbpedia.org/sparql> {
                        <"""+artist_uri+"""> dbp:name ?artist_name .
                        <"""+artist_uri+"""> rdfs:comment ?comment .
                    }
                    FILTER (lang(?comment) = 'fr')
                    }
```

Pour lancer l'application, il faut se placer dans le répertoire **web-app** et exécuter la commande :

```
> flask run
```

Visualisation de l'interface :



Festival City

Enter a city... Submit

or

Enter a region... Submit

A Strasbourg il y a 1 policier pour 1829 habitants

Liste des Festivals

Nom	Artistes
Pelpass Festival	Boys Noize La Femme The Bad Plus The Vaccines
L'Afrique Festival	
Rencontres de l'Illustration