Semantic Web Project

Delivery Food App

Group Mohammed **ROUABAH**, William **MAILLARD**

13/01/2024

Résumé

Table des matières

Introduction

This project aims to collect restaurants information by web scrapping their web-pages on the 'coopcycle.org' domain in order to expose them in a triple store database. These injected data is validated against a shacl validation graph, in order to keep the consistency of the database.

Then, this database is used in our application to provide restaurant's information, given filters set by a user, using a SPARQL query. These filters represent the preferences of the user and can also be saved in the database, formatted as turtle graph.

Therefore in this report we will explain how each of these steps, namely collect - describe - query, has been achieved, before describing the use of the project through the use of the command line or the web server.

Finally we will explain how we deployed and parameterized our own instance of fuseki as well as our flask web application on a PaaS named Heroku.

In order to make the report more readable all technologies and libraries used to make the project are referenced in the annex B with a description about how they have been used.

1 Collect data from Coopcycle

1.1 Scrapping

First of all, coopcycle is a web site that offers delivery service for restaurants. This web site is composed of a web domain at 'coopcycle.org' that link the user to all of the subdomain at '[subdomainname].coopcycle.org' that gather restaurants pages by their location.

Secondly, we notice that restaurants pages have some structures in the class name of their tags that can be used to extract informations about the food delivered by these restaurants.

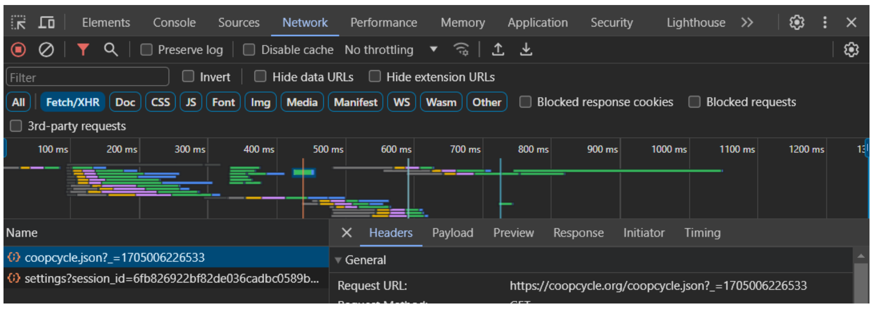
Therefore, to do web scrapping on restaurants websites hosted on the coopcycle.org domain we have been followed these steps :

1. get all subdomains of coopcycle.org,
2. from each subdomain get all restaurants websites home URL
3. from each restaurant website get the jsonld representation of it,
4. from each restaurant website extract restaurant's menus.

1.1.1 subdomains

For getting all subdomains of coopcycle.org we investigate the given URL on the subject. When we land on the page, we notice a map listing all federations registered on coopcycle, so we try to find the file used to build this map.

So we look in the network section using the 'fetch/XHR' filter in order to see all data file used by the site and found the one we wanted called 'coopcycle.json' containing all federations URL.



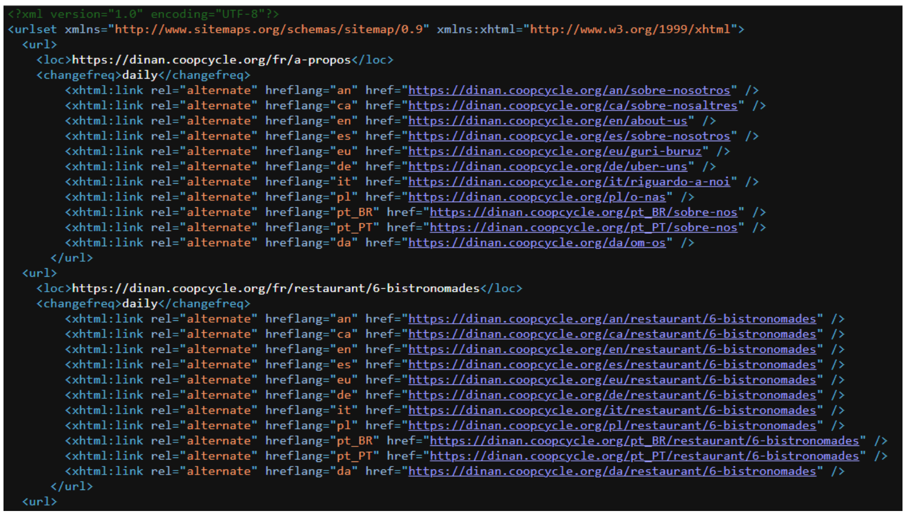
coopcycle.json?_=1705006226533
settings?session_id=6fb826922bf82de036cadbc0589b...

Request URL: https://coopcycle.org/coopcycle.json?_=1705006226533

FIGURE 1 – coopcycle.json fetch request

1.1.2 ld+json

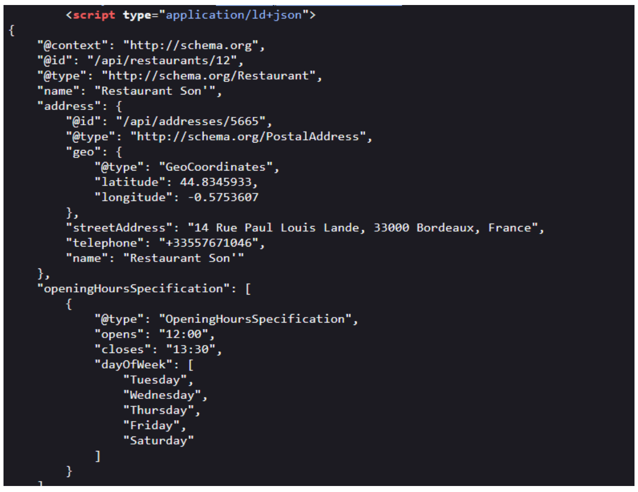
After getting the federations subdomain list, we had to find a way to get restaurants URLS. For that we fetch the sitemap.xml of each federation to acquire its pages URLS and its update frequency.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9" xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <url>
    <loc>https://dinan.coopcycle.org/fr/a-propos</loc>
    <changefreq>daily</changefreq>
        <xhtml:link rel="alternate" hreflang="an" href="https://dinan.coopcycle.org/an/sobre-nosotros" />
        <xhtml:link rel="alternate" hreflang="ca" href="https://dinan.coopcycle.org/ca/sobre-nosaltres" />
        <xhtml:link rel="alternate" hreflang="en" href="https://dinan.coopcycle.org/en/about-us" />
        <xhtml:link rel="alternate" hreflang="es" href="https://dinan.coopcycle.org/es/sobre-nosotros" />
        <xhtml:link rel="alternate" hreflang="eu" href="https://dinan.coopcycle.org/eu/guri-buruz" />
        <xhtml:link rel="alternate" hreflang="de" href="https://dinan.coopcycle.org/de/uber-uns" />
        <xhtml:link rel="alternate" hreflang="it" href="https://dinan.coopcycle.org/it/riguardo-a-noi" />
        <xhtml:link rel="alternate" hreflang="pl" href="https://dinan.coopcycle.org/pl/o-nas" />
        <xhtml:link rel="alternate" hreflang="pt_BR" href="https://dinan.coopcycle.org/pt_BR/sobre-nos" />
        <xhtml:link rel="alternate" hreflang="pt_PT" href="https://dinan.coopcycle.org/pt_PT/sobre-nos" />
        <xhtml:link rel="alternate" hreflang="da" href="https://dinan.coopcycle.org/da/om-os" />
      </url>
  <url>
    <loc>https://dinan.coopcycle.org/fr/restaurant/6-bistronomades</loc>
    <changefreq>daily</changefreq>
        <xhtml:link rel="alternate" hreflang="an" href="https://dinan.coopcycle.org/an/restaurant/6-bistronomades" />
        <xhtml:link rel="alternate" hreflang="ca" href="https://dinan.coopcycle.org/ca/restaurant/6-bistronomades" />
        <xhtml:link rel="alternate" hreflang="en" href="https://dinan.coopcycle.org/en/restaurant/6-bistronomades" />
        <xhtml:link rel="alternate" hreflang="es" href="https://dinan.coopcycle.org/es/restaurant/6-bistronomades" />
        <xhtml:link rel="alternate" hreflang="eu" href="https://dinan.coopcycle.org/eu/restaurant/6-bistronomades" />
        <xhtml:link rel="alternate" hreflang="de" href="https://dinan.coopcycle.org/de/restaurant/6-bistronomades" />
        <xhtml:link rel="alternate" hreflang="it" href="https://dinan.coopcycle.org/it/restaurant/6-bistronomades" />
        <xhtml:link rel="alternate" hreflang="pl" href="https://dinan.coopcycle.org/pl/restaurant/6-bistronomades" />
        <xhtml:link rel="alternate" hreflang="pt_BR" href="https://dinan.coopcycle.org/pt_BR/restaurant/6-bistronomades" />
        <xhtml:link rel="alternate" hreflang="pt_PT" href="https://dinan.coopcycle.org/pt_PT/restaurant/6-bistronomades" />
        <xhtml:link rel="alternate" hreflang="da" href="https://dinan.coopcycle.org/da/restaurant/6-bistronomades" />
      </url>
  <url>
```

FIGURE 2 – content of a sitemap.xml file

Finally, from all gathered URLS we extract the script tag with attribute 'type="application/ld+json"' to obtain the jsonld representation of each restaurant.

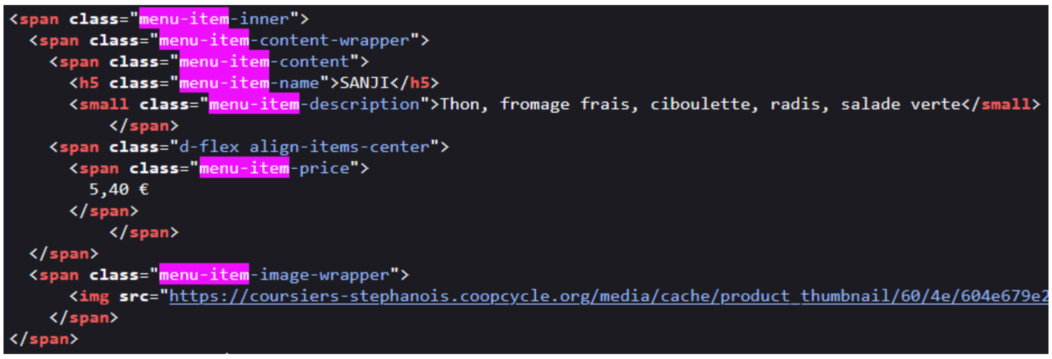


```
        <script type="application/ld+json">
{
    "@context": "http://schema.org",
    "@id": "/api/restaurants/12",
    "@type": "http://schema.org/Restaurant",
    "name": "Restaurant Son'",
    "address": {
        "@id": "/api/addresses/5665",
        "@type": "http://schema.org/PostalAddress",
        "geo": {
            "@type": "GeoCoordinates",
            "latitude": 44.8345933,
            "longitude": -0.5753607
        },
        "streetAddress": "14 Rue Paul Louis Lande, 33000 Bordeaux, France",
        "telephone": "+33557671046",
        "name": "Restaurant Son'"
    },
    "openingHoursSpecification": [
        {
            "@type": "OpeningHoursSpecification",
            "opens": "12:00",
            "closes": "13:30",
            "dayOfWeek": [
                "Tuesday",
                "Wednesday",
                "Thursday",
                "Friday",
                "Saturday"
            ]
        }
```

FIGURE 3 – jsonld illustration sample

1.1.3 Food served

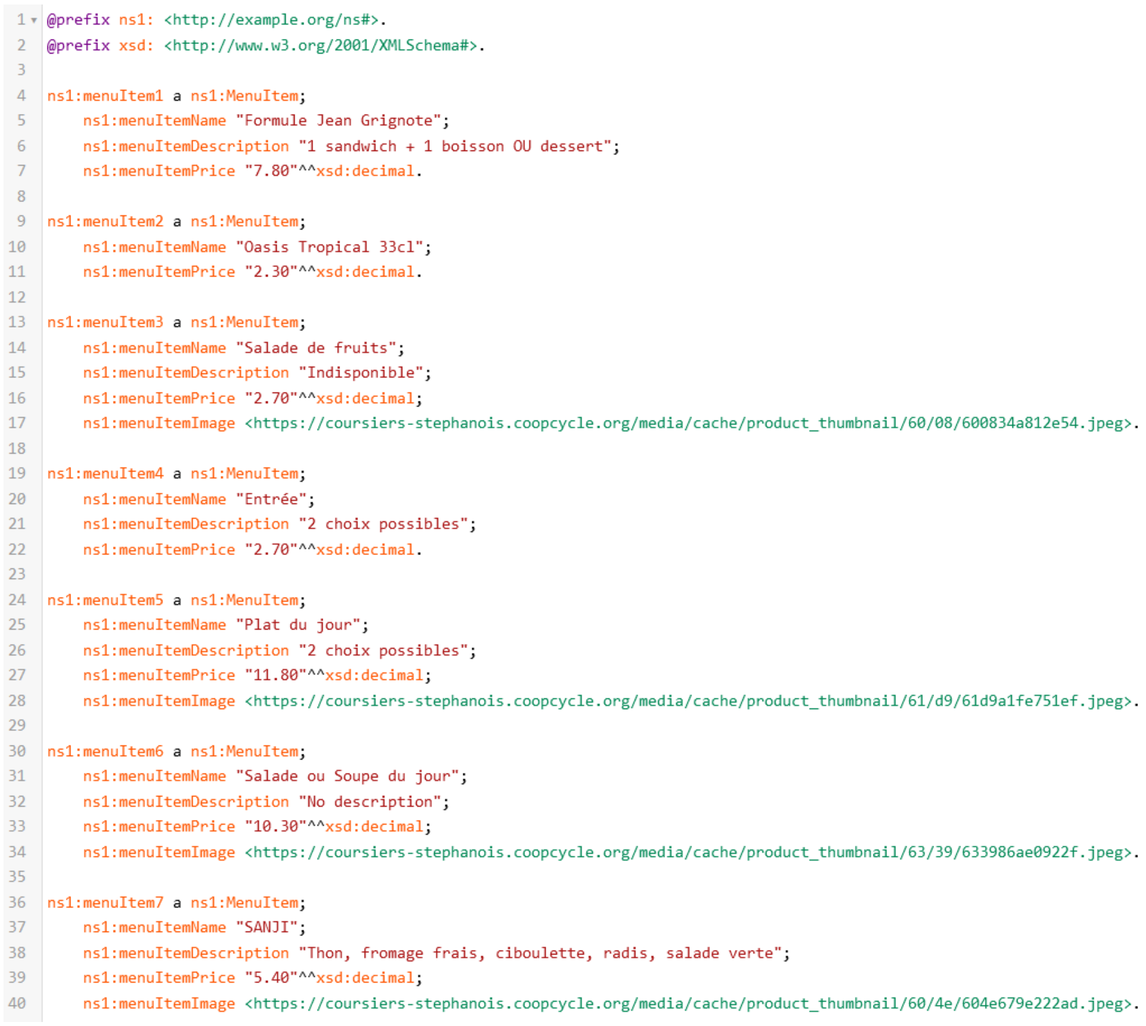
In addition to that we notice a similar structure in restaurants' sites using class name to induce semantic about the information of the tag's content such as the name, the description, the price of the menu and eventually an illustration.



```html
<span class="menu-item-inner">
  <span class="menu-item-content-wrapper">
    <span class="menu-item-content">
      <h5 class="menu-item-name">SANJI</h5>
      <small class="menu-item-description">Thon, fromage frais, ciboulette, radis, salade verte</small>
          </span>
    <span class="d-flex align-items-center">
      <span class="menu-item-price">
        5,40 €
      </span>
          </span>
  </span>
  <span class="menu-item-image-wrapper">
      <img src="https://coursiers-stephanois.coopcycle.org/media/cache/product_thumbnail/60/4e/604e679e2
    </span>
</span>
```

FIGURE 4 – class name semantic for menus

Thus we extract tags, by their class name, related to the information we needed in an intermediate json representation before formatting each of them in our defined turtle structure, using 'schema.org' predicates, to include it inside of the corresponding restaurant graph created from the jsonld (by uploaded it on fuseki, see below).



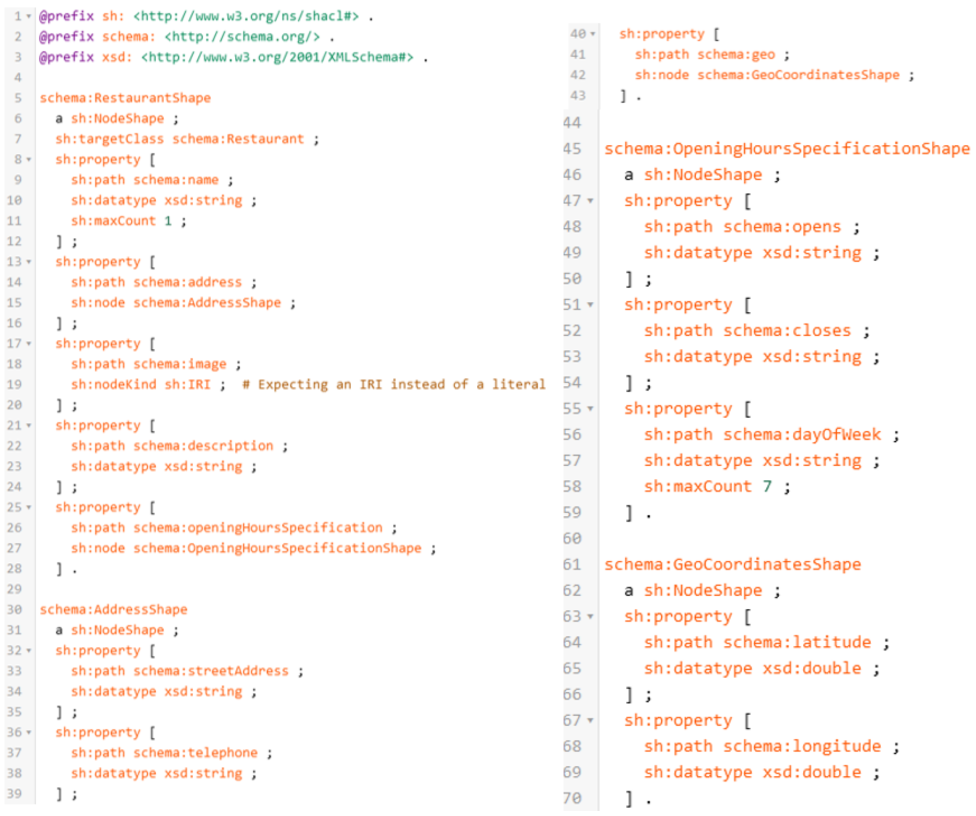
```turtle
@prefix ns1: <http://example.org/ns#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

ns1:menuItem1 a ns1:MenuItem;
    ns1:menuItemName "Formule Jean Grignote";
    ns1:menuItemDescription "1 sandwich + 1 boisson OU dessert";
    ns1:menuItemPrice "7.80"^^xsd:decimal.

ns1:menuItem2 a ns1:MenuItem;
    ns1:menuItemName "Oasis Tropical 33cl";
    ns1:menuItemPrice "2.30"^^xsd:decimal.

ns1:menuItem3 a ns1:MenuItem;
    ns1:menuItemName "Salade de fruits";
    ns1:menuItemDescription "Indisponible";
    ns1:menuItemPrice "2.70"^^xsd:decimal;
    ns1:menuItemImage <https://coursiers-stephanois.coopcycle.org/media/cache/product_thumbnail/60/08/600834a812e54.jpeg>.

ns1:menuItem4 a ns1:MenuItem;
    ns1:menuItemName "Entrée";
    ns1:menuItemDescription "2 choix possibles";
    ns1:menuItemPrice "2.70"^^xsd:decimal.

ns1:menuItem5 a ns1:MenuItem;
    ns1:menuItemName "Plat du jour";
    ns1:menuItemDescription "2 choix possibles";
    ns1:menuItemPrice "11.80"^^xsd:decimal;
    ns1:menuItemImage <https://coursiers-stephanois.coopcycle.org/media/cache/product_thumbnail/61/d9/61d9a1fe751ef.jpeg>.

ns1:menuItem6 a ns1:MenuItem;
    ns1:menuItemName "Salade ou Soupe du jour";
    ns1:menuItemDescription "No description";
    ns1:menuItemPrice "10.30"^^xsd:decimal;
    ns1:menuItemImage <https://coursiers-stephanois.coopcycle.org/media/cache/product_thumbnail/63/39/633986ae0922f.jpeg>.

ns1:menuItem7 a ns1:MenuItem;
    ns1:menuItemName "SANJI";
    ns1:menuItemDescription "Thon, fromage frais, ciboulette, radis, salade verte";
    ns1:menuItemPrice "5.40"^^xsd:decimal;
    ns1:menuItemImage <https://coursiers-stephanois.coopcycle.org/media/cache/product_thumbnail/60/4e/604e679e222ad.jpeg>.
```

FIGURE 5 – class name semantic for menus

1.2 Validating

To make sure the data collected are in a consistent format we created a shacl turtle graph defining rules for a graph that need to be validated so that it can be upload to our triple store database.

Hence, this shacl graph, aims to check that the restaurant graph has some predicates with the correct object's type. Among others, it checks for the existence of a string name, an IRI picture or opening hours of type 'schema :openingHoursSpecification'.



```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix schema: <http://schema.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

schema:RestaurantShape
  a sh:NodeShape ;
  sh:targetClass schema:Restaurant ;
  sh:property [
    sh:path schema:name ;
    sh:datatype xsd:string ;
    sh:maxCount 1 ;
  ] ;
  sh:property [
    sh:path schema:address ;
    sh:node schema:AddressShape ;
  ] ;
  sh:property [
    sh:path schema:image ;
    sh:nodeKind sh:IRI ;  # Expecting an IRI instead of a literal
  ] ;
  sh:property [
    sh:path schema:description ;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
    sh:path schema:openingHoursSpecification ;
    sh:node schema:OpeningHoursSpecificationShape ;
  ] .

schema:AddressShape
  a sh:NodeShape ;
  sh:property [
    sh:path schema:streetAddress ;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
    sh:path schema:telephone ;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
    sh:path schema:geo ;
    sh:node schema:GeoCoordinatesShape ;
  ] .

schema:OpeningHoursSpecificationShape
  a sh:NodeShape ;
  sh:property [
    sh:path schema:opens ;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
    sh:path schema:closes ;
    sh:datatype xsd:string ;
  ] ;
  sh:property [
    sh:path schema:dayOfWeek ;
    sh:datatype xsd:string ;
    sh:maxCount 7 ;
  ] .

schema:GeoCoordinatesShape
  a sh:NodeShape ;
  sh:property [
    sh:path schema:latitude ;
    sh:datatype xsd:double ;
  ] ;
  sh:property [
    sh:path schema:longitude ;
    sh:datatype xsd:double ;
  ] .
```

FIGURE 6 – Restaurant shacl validation graph

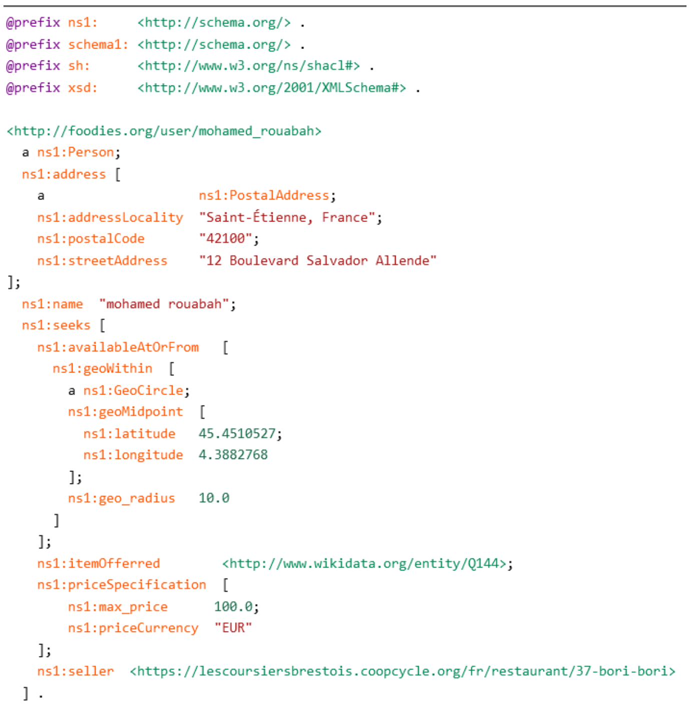
1.3 Injecting

After the validation of the graph we can then send it to fuseki by making a POST request at the endpoint '/data' of our dataset '/foodies' with the GET parameter 'graph' to define the name of the graph and the serialize graph in the body of the request with the corresponding Content-Type header : application/ld+json, text/turtle.

We also parameterise the URL used and the header with environment variables so as to ease the transition between the DEV environment on localhost to the PROD environment on heroku (c.f 5).

2 Describe user preferences

Secondly, in order to return pertinent results to a user we need to gather some information about this user. Therefore we ask the user about its delivery's preferences and then save them has a turtle graph on fuseki to be reused for later.



```
@prefix ns1:     <http://schema.org/> .
@prefix schema1: <http://schema.org/> .
@prefix sh:      <http://www.w3.org/ns/shacl#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .

<http://foodies.org/user/mohamed_rouabah>
  a ns1:Person;
  ns1:address [
    a                    ns1:PostalAddress;
    ns1:addressLocality  "Saint-Étienne, France";
    ns1:postalCode       "42100";
    ns1:streetAddress    "12 Boulevard Salvador Allende"
];
  ns1:name  "mohamed rouabah";
  ns1:seeks [
    ns1:availableAtOrFrom   [
      ns1:geoWithin  [
        a ns1:GeoCircle;
        ns1:geoMidpoint  [
          ns1:latitude   45.4510527;
          ns1:longitude  4.3882768
        ];
        ns1:geo_radius   10.0
      ]
    ];
    ns1:itemOfferred        <http://www.wikidata.org/entity/Q144>;
    ns1:priceSpecification  [
        ns1:max_price      100.0;
        ns1:priceCurrency  "EUR"
    ];
    ns1:seller  <https://lescoursiersbrestois.coopcycle.org/fr/restaurant/37-bori-bori>
  ] .
```

```
       curl https://dsc2-sw-food-delivery-b3a7e3e908fb.herokuapp.com/preferences/?graph= -
           - http://foodies.org/user/mohamed_rouabah
```

FIGURE 7 – Example of preferences turtle graph from

Except from the location, these preferences are optional, nevertheless some are set by default as follow :

- lat & lon are set using the current user position gotten from a geolocation library,
- the opening hours and days parameters are set to the current one,
- the sort key is the distance.

3 Query data from our LDP using SPARQL

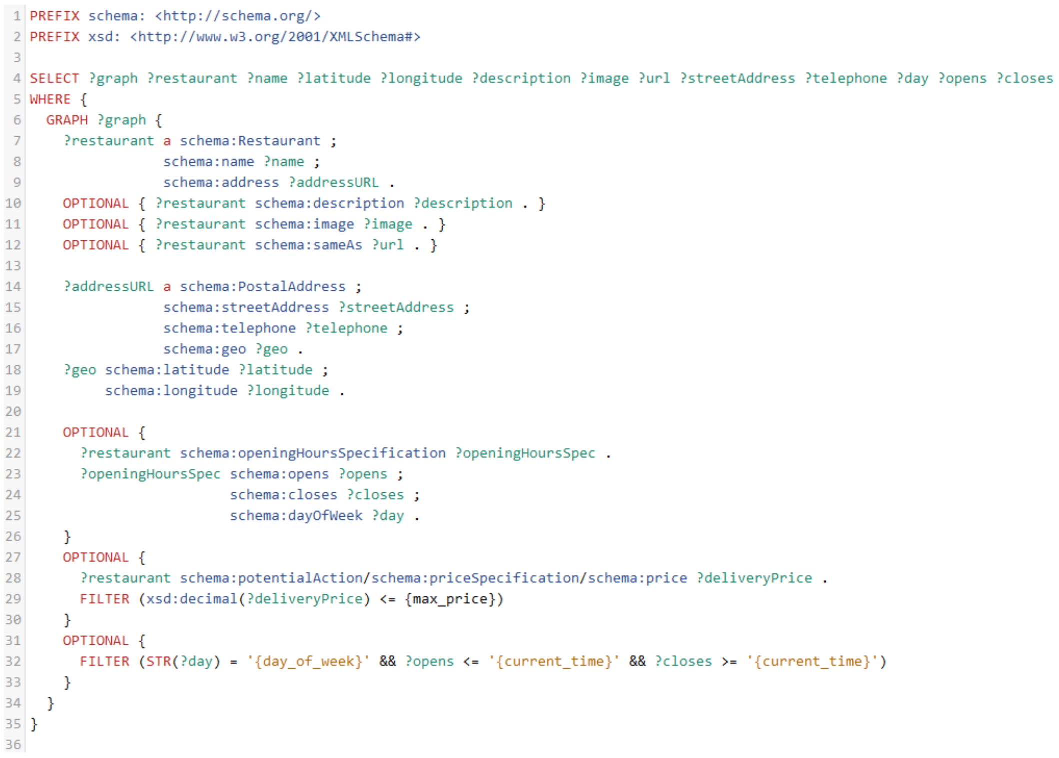
Thirdly, the user's preferences are used to query the triple store database using a parameterized SPARQL request. And a second one is made to get the menus offered by a give restaurant.

3.1 Search for restaurants

Given that some user's preferences are optional we define a flexible SPARQL restaurants request, using **SPARQLWrapper** python module, to adapt of the number of information provided.

Furthermore, we are looking for best suited restaurants for a user even if they not fully match this user's preferences. To that intend, we used the OPTIONAL SPARQL's keyword combine with the FILTER one to get the closest results from the user's preferences.

In addition to that, we parameterize the request with python to not include optional statement when their respective preference is not set.



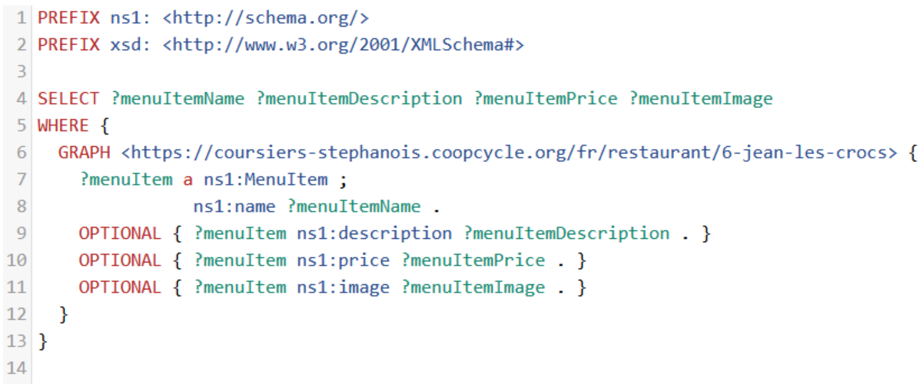
```
PREFIX schema: <http://schema.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?graph ?restaurant ?name ?latitude ?longitude ?description ?image ?url ?streetAddress ?telephone ?day ?opens ?closes
WHERE {
  GRAPH ?graph {
    ?restaurant a schema:Restaurant ;
                schema:name ?name ;
                schema:address ?addressURL .
    OPTIONAL { ?restaurant schema:description ?description . }
    OPTIONAL { ?restaurant schema:image ?image . }
    OPTIONAL { ?restaurant schema:sameAs ?url . }

    ?addressURL a schema:PostalAddress ;
                schema:streetAddress ?streetAddress ;
                schema:telephone ?telephone ;
                schema:geo ?geo .
    ?geo schema:latitude ?latitude ;
         schema:longitude ?longitude .

    OPTIONAL {
      ?restaurant schema:openingHoursSpecification ?openingHoursSpec .
      ?openingHoursSpec schema:opens ?opens ;
                        schema:closes ?closes ;
                        schema:dayOfWeek ?day .
    }
    OPTIONAL {
      ?restaurant schema:potentialAction/schema:priceSpecification/schema:price ?deliveryPrice .
      FILTER (xsd:decimal(?deliveryPrice) <= {max_price})
    }
    OPTIONAL {
      FILTER (STR(?day) = '{day_of_week}' && ?opens <= '{current_time}' && ?closes >= '{current_time}')
    }
  }
}
```

FIGURE 8 – complete SPARQL query to search for restaurants

Then we sort the resulting restaurants by the number of match they have with the user's preferences using the sorted python's function, and by distance or price depending on the one selected by the user.

3.2 Search for food served

After getting the restaurant information, a user can ask for its menu. This is done by a SPARQL query that takes as a parameter the URI of the restaurant, and then searches for object of type 'MenuItem' that belong to the restaurant's graph.



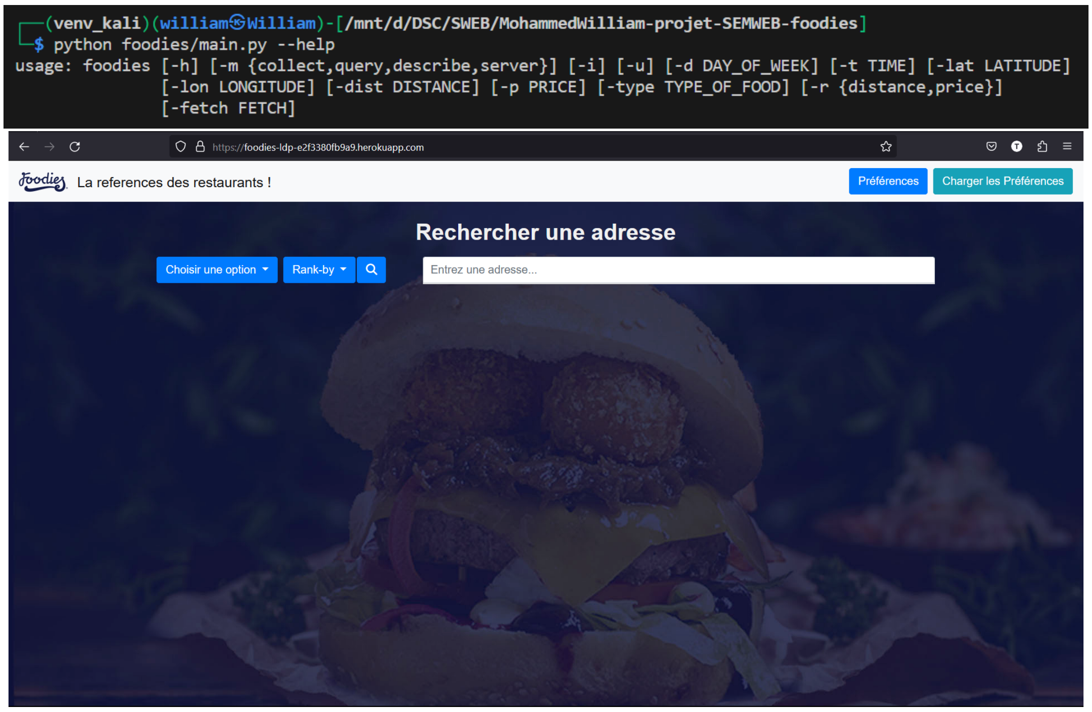
```
PREFIX ns1: <http://schema.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?menuItemName ?menuItemDescription ?menuItemPrice ?menuItemImage
WHERE {
  GRAPH <https://coursiers-stephanois.coopcycle.org/fr/restaurant/6-jean-les-crocs> {
    ?menuItem a ns1:MenuItem ;
               ns1:name ?menuItemName .
    OPTIONAL { ?menuItem ns1:description ?menuItemDescription . }
    OPTIONAL { ?menuItem ns1:price ?menuItemPrice . }
    OPTIONAL { ?menuItem ns1:image ?menuItemImage . }
  }
}
```

FIGURE 9 – SPARQL query to search menus of 'jean les crocs' restaurant

4 Application interfaces

The project contains :

1. A service to collect, describe, query as explain above
2. A CLI using the service
3. A web UI using the service



```
┌──(venv_kali)(william㉿William)-[/mnt/d/DSC/SWEB/MohammedWilliam-projet-SEMWEB-foodies]
└─$ python foodies/main.py --help
usage: foodies [-h] [-m {collect,query,describe,server}] [-i] [-u] [-d DAY_OF_WEEK] [-t TIME] [-lat LATITUDE]
               [-lon LONGITUDE] [-dist DISTANCE] [-p PRICE] [-type TYPE_OF_FOOD] [-r {distance,price}]
               [-fetch FETCH]
```

FIGURE 10 – CLI and WebUI main interface

4.1 Architecture

From the root of the project you will find the following folders and files :

.github/workflows/ CI/CD pipeline :
It is used to analyse the code quality with pylint and automatically deploy the webUI on heroku.

deploy/fuseki/ Dockerfile, script bash, configuration files :
It is used to build, configure and deploy a jena-fuseki container on heroku.

foodies/ source code of the application :

controllers/ :
define the routes of HTTP requests of the project service.

coopcycle_scrapper/ :
define the classes used to scrap data from coopcycle.org, validate it, and send it to fuseki.

models/ :

describe.py : methods to collect, validate, transform and upload user's preferences.
menu.py : methods to create the menu graph and upload it on fuseki.
query.py : contains the SPARQL queries.

static/ : contains the static files for the web UI.

templates/ : contains the flask html templates for the web UI.

app.py, cache.py : define and configure the flask web app.

config.py : defines project parameters using environment variables.

foodies_parser : creates the CLI argparser.

main.py : used to launch the CLI interface.

modes.py : wrappers for the CLI modes : collect, describe, query.

Procefile heroku configuration file to launch the app.

README.md installtion guidelines.

requirements.txt list of project's dependencies.

4.2 Set-up

The project need python version 3.7 or higher and a venv environment is advice, but not mandatory, to use it. Bellow you can find the following commands to set up the pythons dependencies in order to launch the project.

1. Create a venv named venv_kali
```
python3 -m venv venv_kali
```

2. Activate the created venv
```
source venv_kali/bin/activate
```

3. Install the dependencies
```
pip -r requirements.txt
```

4. Display the cli help
```
python foodies/main.py --help
```

4.3 Command Line Interface

The CLI has been made using the argparse module and has four mode that can be chose with the parameter '-m' or '–mode' :

collect : is used for initialize, scrap coopcycle and populate the triple store database.
query : is used to query the triple store database
describe : is used to gather user's preferences and upload them to jena
server : is used to launch the flask server to consult the UI at 'localhost :5000'

4.3.1 Collect

1. initialize fuseki
```
python foodies/main.py -m collect -i
```

2. Scrap coopcycle website and save data collected in data/

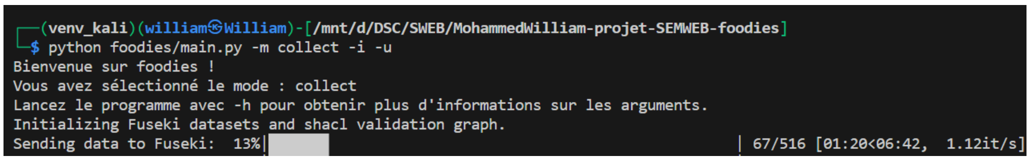
```
python foodies/main.py -m collect
```

3. Upload scrapped data from data/

```
python foodies/main.py -m collect -u
```

Commands 1-3 combine

```
python foodies/main.py -m collect -i -u
```

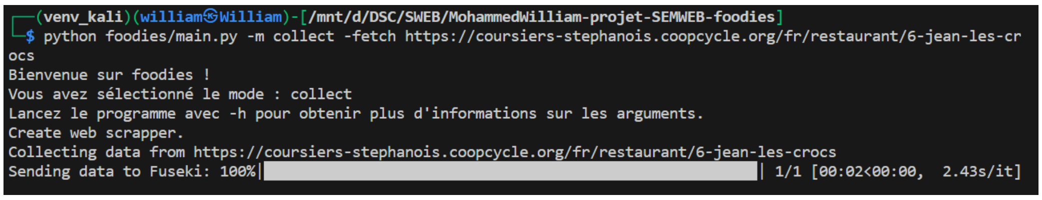


```
┌──(venv_kali)(william㉿William)-[/mnt/d/DSC/SWEB/MohammedWilliam-projet-SEMWEB-foodies]
└─$ python foodies/main.py -m collect -i -u
Bienvenue sur foodies !
Vous avez sélectionné le mode : collect
Lancez le programme avec -h pour obtenir plus d'informations sur les arguments.
Initializing Fuseki datasets and shacl validation graph.
Sending data to Fuseki:  13%|██                                   | 67/516 [01:20<06:42,  1.12it/s]
```

FIGURE 11 – collect mode

4. Scrap a given url

```
python foodies/main.py -m collect -fetch
↪ https://coursiers-stephanois.coopcycle.org/fr/restaurant/6-jean-les-crocs
```



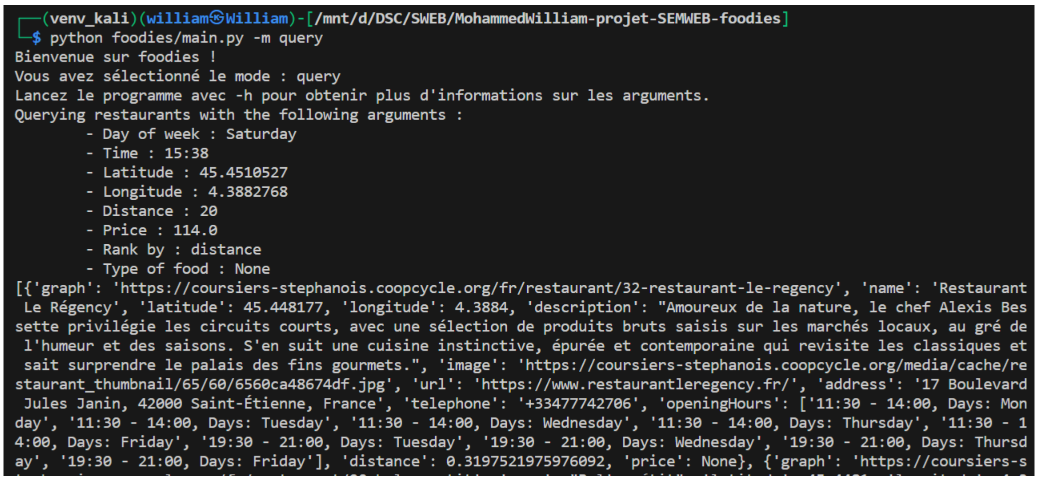
```
┌──(venv_kali)(william㉿William)-[/mnt/d/DSC/SWEB/MohammedWilliam-projet-SEMWEB-foodies]
└─$ python foodies/main.py -m collect -fetch https://coursiers-stephanois.coopcycle.org/fr/restaurant/6-jean-les-cr
ocs
Bienvenue sur foodies !
Vous avez sélectionné le mode : collect
Lancez le programme avec -h pour obtenir plus d'informations sur les arguments.
Create web scrapper.
Collecting data from https://coursiers-stephanois.coopcycle.org/fr/restaurant/6-jean-les-crocs
Sending data to Fuseki: 100%|██████████████████████████████████████████| 1/1 [00:02<00:00,  2.43s/it]
```

FIGURE 12 – collect mode with fetch argument

4.3.2 Query

Query with default values

```
python foodies/main.py -m query
```

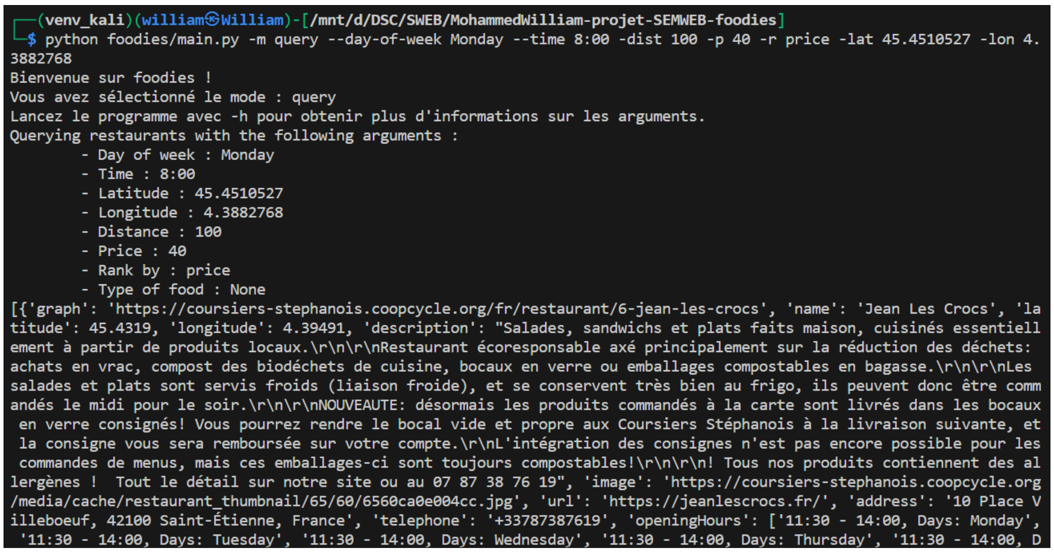


```
┌──(venv_kali)(william㉿William)-[/mnt/d/DSC/SWEB/MohammedWilliam-projet-SEMWEB-foodies]
└─$ python foodies/main.py -m query
Bienvenue sur foodies !
Vous avez sélectionné le mode : query
Lancez le programme avec -h pour obtenir plus d'informations sur les arguments.
Querying restaurants with the following arguments :
        - Day of week : Saturday
        - Time : 15:38
        - Latitude : 45.4510527
        - Longitude : 4.3882768
        - Distance : 20
        - Price : 114.0
        - Rank by : distance
        - Type of food : None
[{'graph': 'https://coursiers-stephanois.coopcycle.org/fr/restaurant/32-restaurant-le-regency', 'name': 'Restaurant
 Le Régency', 'latitude': 45.448177, 'longitude': 4.3884, 'description': "Amoureux de la nature, le chef Alexis Bes
sette privilégie les circuits courts, avec une sélection de produits bruts saisis sur les marchés locaux, au gré de
 l'humeur et des saisons. S'en suit une cuisine instinctive, épurée et contemporaine qui revisite les classiques et
 sait surprendre le palais des fins gourmets.", 'image': 'https://coursiers-stephanois.coopcycle.org/media/cache/re
staurant_thumbnail/65/60/6560ca48674df.jpg', 'url': 'https://www.restaurantleregency.fr/', 'address': '17 Boulevard
 Jules Janin, 42000 Saint-Étienne, France', 'telephone': '+33477742706', 'openingHours': ['11:30 - 14:00, Days: Mon
day', '11:30 - 14:00, Days: Tuesday', '11:30 - 14:00, Days: Wednesday', '11:30 - 14:00, Days: Thursday', '11:30 - 1
4:00, Days: Friday', '19:30 - 21:00, Days: Tuesday', '19:30 - 21:00, Days: Wednesday', '19:30 - 21:00, Days: Thursd
ay', '19:30 - 21:00, Days: Friday'], 'distance': 0.3197521975976092, 'price': None}, {'graph': 'https://coursiers-s
```

FIGURE 13 – query mode

Query with specific values

```
python foodies/main.py -m query --day-of-week Monday --time 8:00 -dist 100 -p 40 -r
↪ price -lat 45.4510527 -lon 4.3882768
```



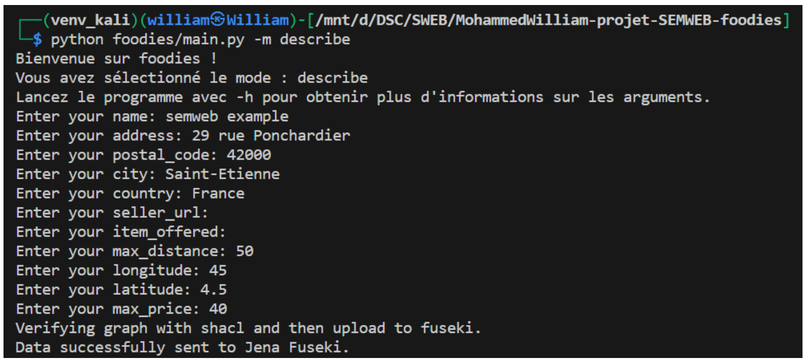
```
┌─(venv_kali)(william㉿William)-[/mnt/d/DSC/SWEB/MohammedWilliam-projet-SEMWEB-foodies]
└─$ python foodies/main.py -m query --day-of-week Monday --time 8:00 -dist 100 -p 40 -r price -lat 45.4510527 -lon 4.
3882768
Bienvenue sur foodies !
Vous avez sélectionné le mode : query
Lancez le programme avec -h pour obtenir plus d'informations sur les arguments.
Querying restaurants with the following arguments :
        - Day of week : Monday
        - Time : 8:00
        - Latitude : 45.4510527
        - Longitude : 4.3882768
        - Distance : 100
        - Price : 40
        - Rank by : price
        - Type of food : None
[{'graph': 'https://coursiers-stephanois.coopcycle.org/fr/restaurant/6-jean-les-crocs', 'name': 'Jean Les Crocs', 'la
titude': 45.4319, 'longitude': 4.39491, 'description': "Salades, sandwichs et plats faits maison, cuisinés essentiell
ement à partir de produits locaux.\r\n\r\nRestaurant écoresponsable axé principalement sur la réduction des déchets:
achats en vrac, compost des biodéchets de cuisine, bocaux en verre ou emballages compostables en bagasse.\r\n\r\nLes
salades et plats sont servis froids (liaison froide), et se conservent très bien au frigo, ils peuvent donc être comm
andés le midi pour le soir.\r\n\r\nNOUVEAUTE: désormais les produits commandés à la carte sont livrés dans les bocaux
 en verre consignés! Vous pourrez rendre le bocal vide et propre aux Coursiers Stéphanois à la livraison suivante, et
 la consigne vous sera remboursée sur votre compte.\r\nL'intégration des consignes n'est pas encore possible pour les
 commandes de menus, mais ces emballages-ci sont toujours compostables!\r\n\r\n! Tous nos produits contiennent des al
lergènes !  Tout le détail sur notre site ou au 07 87 38 76 19", 'image': 'https://coursiers-stephanois.coopcycle.org
/media/cache/restaurant_thumbnail/65/60/6560ca0e004cc.jpg', 'url': 'https://jeanlescrocs.fr/', 'address': '10 Place V
illeboeuf, 42100 Saint-Étienne, France', 'telephone': '+33787387619', 'openingHours': ['11:30 - 14:00, Days: Monday',
 '11:30 - 14:00, Days: Tuesday', '11:30 - 14:00, Days: Wednesday', '11:30 - 14:00, Days: Thursday', '11:30 - 14:00, D
```

FIGURE 14 – query mode

4.3.3 Describe

Create preferences graph and upload to fuseki

```
python foodies/main.py -m describe
```

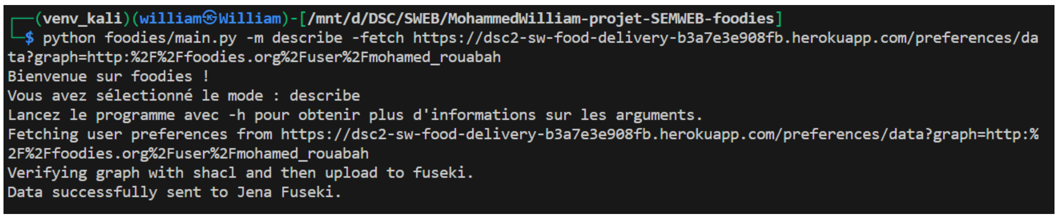


```
┌─(venv_kali)(william㉿William)-[/mnt/d/DSC/SWEB/MohammedWilliam-projet-SEMWEB-foodies]
└─$ python foodies/main.py -m describe
Bienvenue sur foodies !
Vous avez sélectionné le mode : describe
Lancez le programme avec -h pour obtenir plus d'informations sur les arguments.
Enter your name: semweb example
Enter your address: 29 rue Ponchardier
Enter your postal_code: 42000
Enter your city: Saint-Etienne
Enter your country: France
Enter your seller_url:
Enter your item_offered:
Enter your max_distance: 50
Enter your longitude: 45
Enter your latitude: 4.5
Enter your max_price: 40
Verifying graph with shacl and then upload to fuseki.
Data successfully sent to Jena Fuseki.
```

FIGURE 15 – describe mode

Upload preference graph to feuseki given a url

```
python foodies/main.py -m describe -fetch http://localhost:3030/preferences/data?graph=
http:%2F%2Ffoodies.org%2Fuser%2Fmohamed_rouabah
```



```
┌─(venv_kali)(william㉿William)-[/mnt/d/DSC/SWEB/MohammedWilliam-projet-SEMWEB-foodies]
└─$ python foodies/main.py -m describe -fetch https://dsc2-sw-food-delivery-b3a7e3e908fb.herokuapp.com/preferences/da
ta?graph=http:%2F%2Ffoodies.org%2Fuser%2Fmohamed_rouabah
Bienvenue sur foodies !
Vous avez sélectionné le mode : describe
Lancez le programme avec -h pour obtenir plus d'informations sur les arguments.
Fetching user preferences from https://dsc2-sw-food-delivery-b3a7e3e908fb.herokuapp.com/preferences/data?graph=http:%
2F%2Ffoodies.org%2Fuser%2Fmohamed_rouabah
Verifying graph with shacl and then upload to fuseki.
Data successfully sent to Jena Fuseki.
```

FIGURE 16 – describe mode with fetch argument

4.4 Web interface

4.4.1 Architecture of the website

The website is sub divided in three parts.

Toolkit page : it gives the possibilities to collect data, send data and initialize the Linked Data Platform.

Main page : it includes all the main functionalities of the application.

Restaurant page : this is the result page of a query, that page include all the listed restaurants for the current query applied on.

4.4.2 Toolkit page

From that page the dev user can set up the collected data from a single click, the data is collected and sent to our deployed fuseki instance.

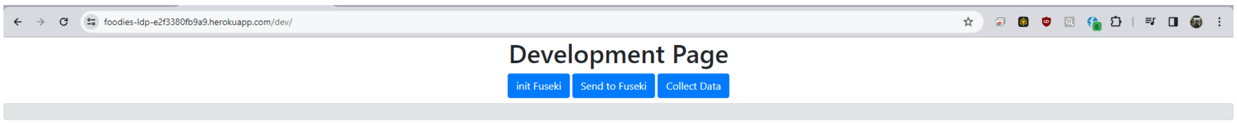
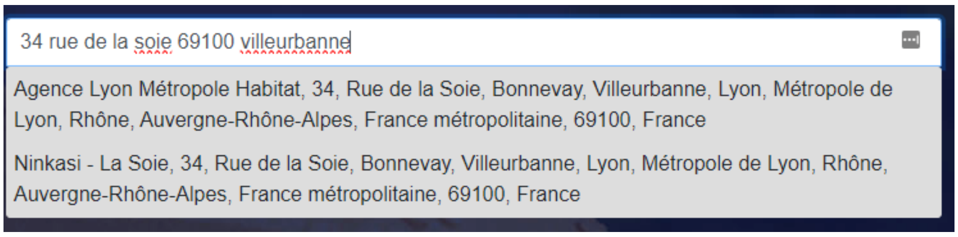


FIGURE 17 – Toolkit page

4.4.3 Main Page

The main page includes the search bar, that use the OpenStreetMap service : https://nominatim.openstreetmap.org/ui/search.html, that service provide us the localization of each postal address.



34 rue de la soie 69100 villeurbanne

Agence Lyon Métropole Habitat, 34, Rue de la Soie, Bonnevay, Villeurbanne, Lyon, Métropole de Lyon, Rhône, Auvergne-Rhône-Alpes, France métropolitaine, 69100, France

Ninkasi - La Soie, 34, Rue de la Soie, Bonnevay, Villeurbanne, Lyon, Métropole de Lyon, Rhône, Auvergne-Rhône-Alpes, France métropolitaine, 69100, France

FIGURE 18 – search bar

From that page we also have a filtered drop-down search bar, which use the current user localization provided by a cookie, using the geocoding API from the navigator.

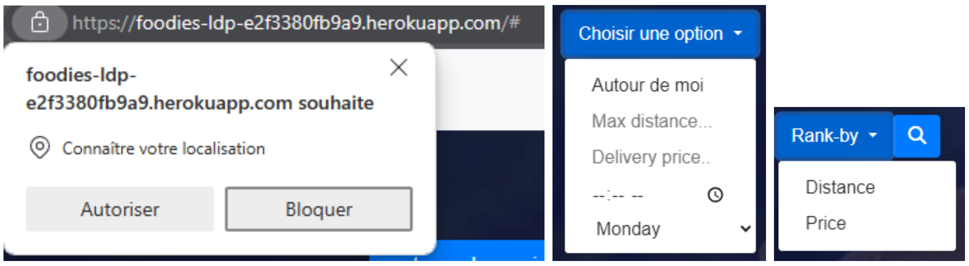
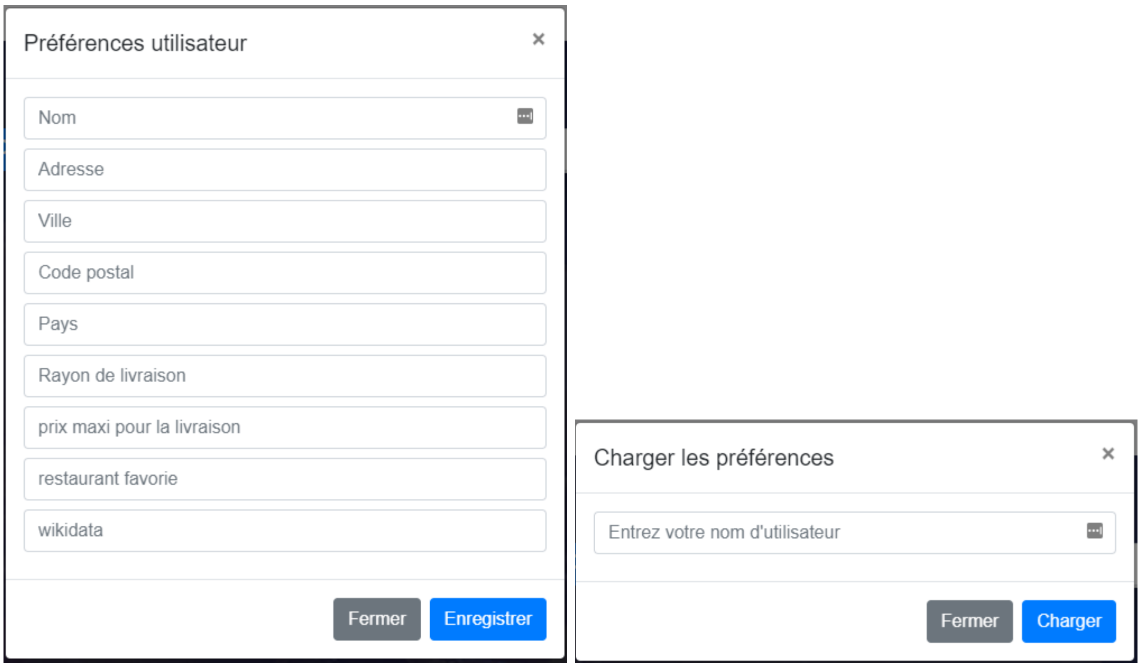

https://foodies-ldp-e2f3380fb9a9.herokuapp.com/#
foodies-ldp-e2f3380fb9a9.herokuapp.com souhaite
Connaître votre localisation
Autoriser
Bloquer
Choisir une option
Autour de moi
Max distance...
Delivery price..
--:-- --
Monday
Rank-by
Distance
Price

FIGURE 19 – filtered search bar

The drop-down include, the distance between the user position and restaurants, the delivery price, the hours and days of the delivery.

Also in the header of the page there is two buttons which give the ability to provide preferences that are added to fuseki, so later on they can be re-used by providing the given name.


Préférences utilisateur ×

Nom

Adresse

Ville

Code postal

Pays

Rayon de livraison

prix maxi pour la livraison

restaurant favorie

wikidata

Fermer Enregistrer

Charger les préférences ×

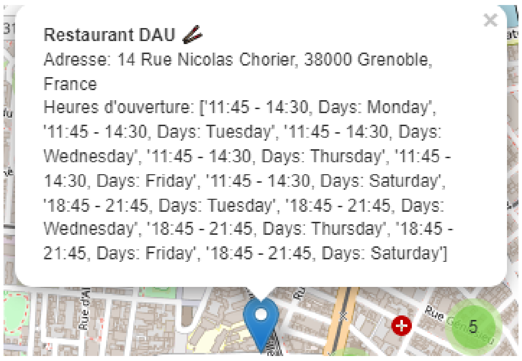
Entrez votre nom d'utilisateur

Fermer Charger

FIGURE 20 – user preferences

4.4.4 Restaurants page

From that page, restaurants' result from the query are displayed. Each restaurant displayed has its own target point inside the Open Search Map by using the latitude and longitude that we got from the triple store.

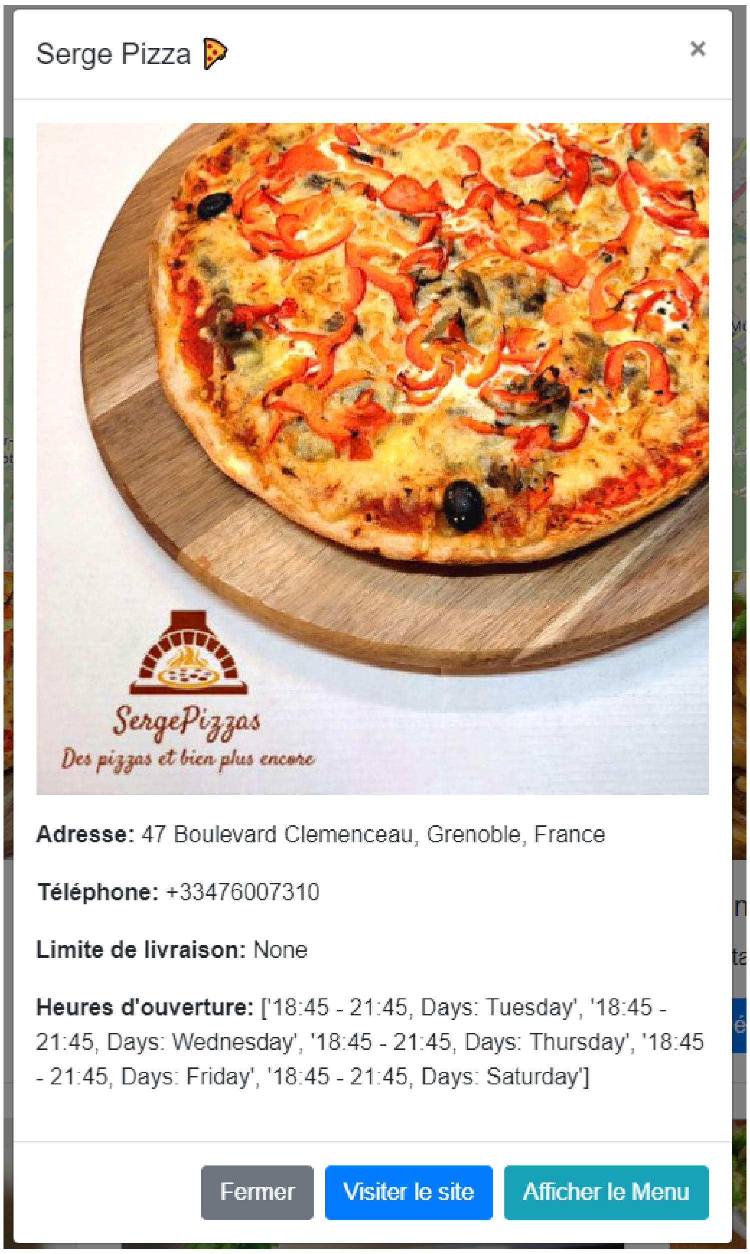

Restaurant DAU

Adresse: 14 Rue Nicolas Chorier, 38000 Grenoble, France

Heures d'ouverture: ['11:45 - 14:30, Days: Monday', '11:45 - 14:30, Days: Tuesday', '11:45 - 14:30, Days: Wednesday', '11:45 - 14:30, Days: Thursday', '11:45 - 14:30, Days: Friday', '11:45 - 14:30, Days: Saturday', '18:45 - 21:45, Days: Tuesday', '18:45 - 21:45, Days: Wednesday', '18:45 - 21:45, Days: Thursday', '18:45 - 21:45, Days: Friday', '18:45 - 21:45, Days: Saturday']

FIGURE 21 – restaurant's tooltip on the map

Also from that page we have details of each restaurant, like their url, name, description, images.


Serge Pizza

Adresse: 47 Boulevard Clemenceau, Grenoble, France

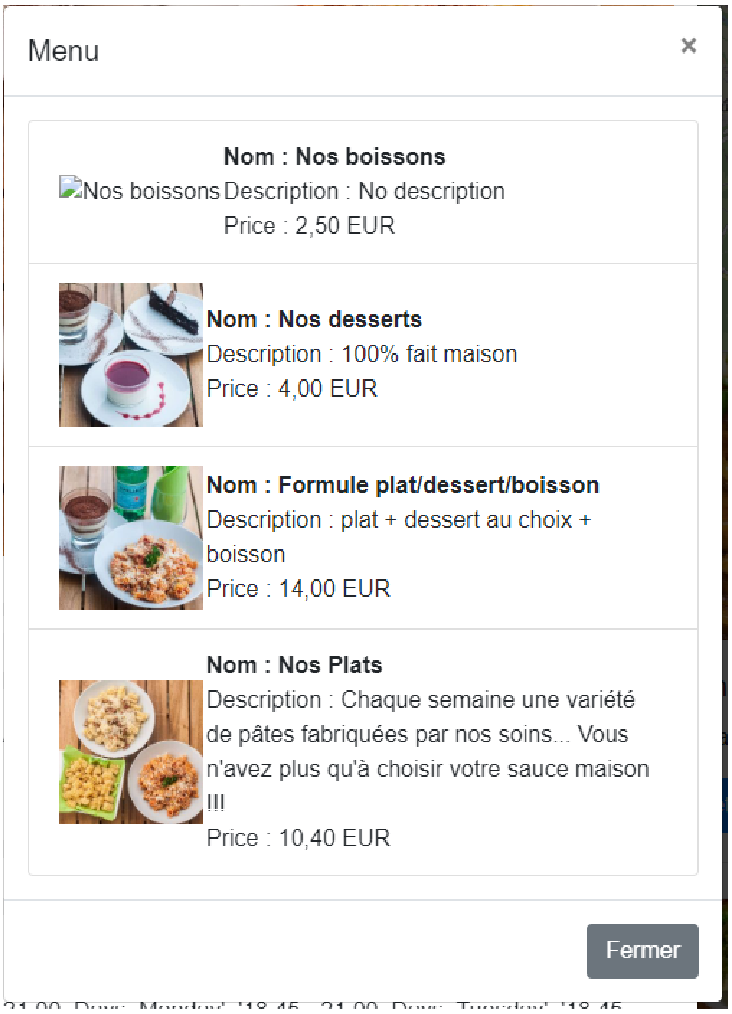
Téléphone: +33476007310

Limite de livraison: None

Heures d'ouverture: ['18:45 - 21:45, Days: Tuesday', '18:45 - 21:45, Days: Wednesday', '18:45 - 21:45, Days: Thursday', '18:45 - 21:45, Days: Friday', '18:45 - 21:45, Days: Saturday']

FIGURE 22 – restaurants information

Additionally, the menu of each restaurants can be displayed by clicking on the 'Afficher Menu' button, if the currently selected restaurant has one.



Menu

×

Nom : Nos boissons
Nos boissons Description : No description
Price : 2,50 EUR

Nom : Nos desserts
Description : 100% fait maison
Price : 4,00 EUR

Nom : Formule plat/dessert/boisson
Description : plat + dessert au choix + boisson
Price : 14,00 EUR

Nom : Nos Plats
Description : Chaque semaine une variété de pâtes fabriquées par nos soins... Vous n'avez plus qu'à choisir votre sauce maison !!!
Price : 10,40 EUR

Fermer

FIGURE 23 – restaurant menu tab

5 Deployment

In order to make our project availible on the internet we deployed on heroku a fuseki database at https://dsc2-sw-food-delivery-b3a7e3e908fb.herokuapp.com/#/ and a flask web UI at https://foodies-ldp-e2f3380fb9a9.herokuapp.com/. Since Heroku app are automatically put to sleep after 15 minutes make sure to wake up the fuseki data base by clicking on its link before trying to use the web UI.

5.1 Jena-fuseki

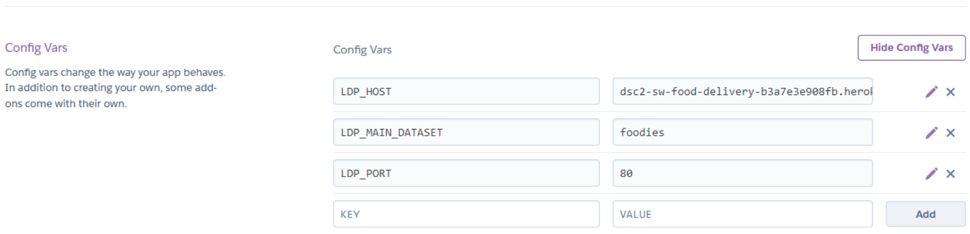
We create a docker container based on the alpine one with jena-fuseki install and our 2 datasets 'foodies' and 'preferences' already populated to make sure data are availible at the start of the container.

We have also modifying the default configuration from 'shiro.ini' in order to make throught a public IP address.

Finally the Heroku CLI has been used to push the container image on Heroku and release the container.

5.2 Web server Flask

Do deploy the Flask application, we create a Procfile that is used by Heroku to launch it and use a GitHub action in our CI/CD pipeline to automatically deploy it. We also make sure to parameterize the configuration of the application with environment variable to have no difference when launching the app on different host.



Config Vars

Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

Config Vars

KEY	VALUE
LDP_HOST	dsc2-sw-food-delivery-b3a7e3e908fb.herol
LDP_MAIN_DATASET	foodies
LDP_PORT	80

FIGURE 24 – Heroku environment variables used to configure the application

A Vocabulary

Vocabulaire	**Définition**
CLI	Command Line Interface is text-based interface used for interacting with a program.
CI/CD	Continuous Integration and Continuous Deployement use to automates the live cycle of an application.
PaaS	Platform as a Service that allows developers to deploy, manage and scale applications without dealing with the underlying infrastructure.
RDF	Ressource Description Framework is a framework to represent information in a form of triple : subject - predicat - object.
robots.txt	File at the root of a web domain to define policy for web crawlers, meaning allow or disallow specific web crawler or URL.
sitemap.xml	File at the root of a web domain that list all website URL and some meta-data like the frequence of updates, in order to help web crawlers crawls the website.
webUI	web User Interface : design the part of a website that interact with the user, also called front-end.

B Technologies used

SPARQL server jena-fuseki 4.10.0 :
self-hosted version on localhost and docker version on heroku.

Programming language python 3.11.5

package manager pip 23.3.2 :
it is used with a Virtual Environment also called venv.

CI/CD github workflow :
it is used to control code quality with pylint and deploying the webapp on heroku.

CLI tool argparse :
built-in Python module to parse the command line arguments to make the CLI.

web framework flask 3.0.0 :
it is used to make the web interface.

web server Gunicorn 21.2.0 :
it is used to launch python web application.

PaaS heroku :
it is used to deploy the web interface and a fuseki container on the internet.

scrapping library Beautiful Soup 4.12.2 :
it is used to parse html pages to scrap ld+json and menus from restaurant pages.

geocoding library Geopy 2.4.1 :
it is used to compute latitude and longitude given and address.

map API OpenStreetMap : it is used to display the results on a map.

shacl library pyshacl 0.25.0 :
use to validate our RDF graphs.

RDF library rdflib 7.0.0 :
it is used to create turtle graph.

HTTP library requests 2.31.0 :
it is used to make http requests.

SPARQL library SPARQLWrapper 2.0.0 :
it is used to create, parameterize and send SPARQL queries.

progress bar library tqdm 4.66.1 :
it is used to visualize the progress of a task on the terminal.

RDF file format turtle :
it is used to make shacl and user's preferences graphs.

C References

- semantic web https://www.emse.fr/~zimmermann/Teaching/SemWeb/
- shiro.ini configuration http://greycode.github.io/shiro/doc/web.html, https://shiro.apache.org/configuration.html
- pip documentation for python modules used https://pip.pypa.io/en/stable/
- for handling geolocation data https://geopy.readthedocs.io/en/stable/