QOTB Badr
SAND Jonathan
TALEB Sammy

WEB DATA MINING PROJECT REPORT

The main objectives of this project were to create a web application connected to Fuseli Triple Stores which allow a user to find nearby interesting places by querying these Triple Stores. Having greater experiences and skills in Python than in Java, we decided to develop the web app in Python language and to connect to the Fuseki server using pyfuseki and rdflib libraries. In this report we will explain the choices we made to produce a web app as complete as possible. The source code of the project can be found at https://github.com/Jonathan2021/semantic_bikes.

PROTÉGÉ ONTOLOGY

We have created the ontology on the protégé software. Each class and properties have been created. We have created several instances. However, as the number of instances is very huge, we have not created all of them (more than 30 000).

The protégé owl file can be found in the Data folder.

REQUIREMENTS

The project has been developed in Python and is connected to the Fuseki API. Python with libraries pyfuseki, rdflib, numpy, json and flask have to be installed. You also have to install the fuseki-server triplestore manager.

RUN THE PROJECT

To run the project, you have to run the flask project and the fuseki server.

RUNNING FLASK SERVER

To run the flask server, move to the folder WebApp Flask and run the command line “python app.py” in a python terminal. The server will be running locally, using the port 5000. (http://127.0.0.1:5000)

RUNNING FUSEKI SERVER

For seek of simplicity and to be able to run update HTTP queries, we have pushed on Github the whole database in the folder Data/tripleStore. You can also find the turtle file that we have used to create this database in the folder Data.

To run the Apache Fuseki server, you’ll launch the server, allowing update queries on the database folder location using the following command line:

“fuseki-server - -loc=”YOUR_PATH/semantic_bikes/Data/tripleStore” - - update /database”

Note that YOUR_PATH is the folder where you have pulled the Github repository. Note also that it is very important to use the parameter - - update and the same database name (/database) to be able to run all HTTP queries properly.

If you are using a Linux OS, we have created a MakeFile which will launch all of these command lines.

QOTB Badr
SAND Jonathan
TALEB Sammy

LIBRARIES & POST OFFICES SYSTEMS

ONTOLOGIES

The first two ontologies we decided to create are libraries and a post offices systems. To do so, we've used large CSVs referencing all libraries and all post offices in France. As the data is static, we have first created a small script in Python to transform the CSV files into two Ontologies saved in a Turtle format. We then used this turtle format to populate two Triple Stores in Fuseki named libraries and postoffices in our project. As there is no need to update these Triple Stores, the data is persistent in the database and the script transforming the CSV into Turtle files is not integrated inside the main project code.

WEB APP PART

Concerning the web app part, it works as following. A user just has to fill in a Form with his address, Zip code and city and a maximal distance where he wants the libraries to be found (3km, 5km, 10km, 20km, No limit). The web app will then display all libraries/post offices found in a distance smaller than the maximal distance selected.

The main problem here was to find a way to calculate distances between, two addresses. In the ontologies, we had the geocoordinates, but the user is simply giving his address. We had to find a way to transform this address into geocoordinates. To do so, we used an API named api.positionstack.com which can be requested by creating a link with specific parameters (address, …). This API returns then a JSON file containing the geocoordinates. We Finally had to use the spheric coordinates to compute the distance between two points.

Concerning querying the Triple Stores, we have constructed simple SAPRQL Queries to get information about libraries / post offices by simply filtering them by Departement (two first numbers of the Zip Code). The pyfuseki has built-in function which allows to connect to the server and the perform efficient queries.

To Summary, when a user is giving his address, we get the geocoordinates associated to this address and the web app returns the libraries/post offices around him ordered from the closest to the furthest.

The final interesting thing is that we've integrated RDFa inside the final page in order to be able to query the web page with the Google Testing Tool, for instance. This way, we would be able to retrieve information about any Library System / Post Office System by simply querying the HTML source code of the page.

QOTB Badr
SAND Jonathan
TALEB Sammy

BIKE SHARING SYSTEM

ONTOLOGY

The other ontology we decided to integrate within the project is the bicycle sharing system. As the number of bikes available and the number of empty places are changing in a specific station, the data is dynamic. To handle this, we have created a complete module in Python which is used to scrap data from CSV/JSON files on the net and to transform them into RDF ontologies using RDFlib. Finally, we transform them into Turtle files and add them to the fuseki Triple Stores in order to update them.

WEB APP PART

Concerning the web app part, it works as following. A user just has to fill in a Form with his start address, final address and a city. The Web app only allows a starting point and a final destination within the same city as the bike sharing system may be different in two different cities. After that the first task done by the web app is to update the triple store for this city. To do so, we just perform a delete data and an insert data SPARQL queries with the new data parsed by the python module, and all filtered by a specific city. Until now, the web app handles 17 different bike sharing systems (17 cities) scrapped from 17 different source files on the web. The cities are:

- Amien
- Avignon
- Besancon
- Cergy-Pontoise
- Créteil
- Lyon
- Marseille
- Mulhouse
- Montpellier
- Nancy
- Nantes
- Nice
- Renne
- Rouen
- Strasbourg
- Toulouse
- Valence

After performing the queries, we simply used a select query to find all bike sharing system in a city. We use the same API to find the geocoordinates associated to the start and destination addresses. We then find the closest bike sharing system from the start point (with at least one bike available) and to the destination point (with at least one free place). The itinerary is simply written with all distances (walk distance + ride distance) which allow the user to know his itinerary.

If no bike sharing system with free places or available bikes is found, or if the distance is too high (for instance, 1 km between the two destination and 6km rides if we use bike station), the web app simply returns that no bike sharing system have been found and the user has to walk.

The final interesting thing is that we've integrated RDFa inside the final page in order to be able to query the web page with the Google Testing Tool, for instance. This way, we would be able to retrieve information about any bike sharing System by simply querying the HTML source code of the page.