Bases de données avancées GLO-7035
Parcours de vélo épicurien

Adrien TURCHINI
Marc HENRIOT

27 octobre 2020

Table des matières

1 Source des données **2**
1.1 Sources 2
1.2 Exemple de données 2

2 Technologies utilisées **2**
2.1 Language de programmation : Javascript 2
2.2 Bases de données 2

3 Processus d'ETL **3**
3.1 Processus d'acquitision initiales des données 3
3.2 Processus d'acquisition incrémental de données 3
3.3 Processus de transformation des données transformées 3
3.3.1 MongoDb et TripAdvisor 3
3.3.2 Neo4j et les pistes cyclables 4
3.4 Schéma de la pipeline d'ETL 5

4 Annexe **6**
4.1 Exemple de données de restaurant 6
4.2 Exemple de données de piste cyclable 6
4.3 Exemple de données de restaurant après scraping 6
4.4 Exemple de données de piste cyclable 7

5 Remarques **8**

1 Source des données

1.1 Sources

Nous avons choisi la ville de Québec afin de réaliser notre projet. L'Université Laval se trouvant dans cette ville qui par ailleurs est de taille assez grande pour avoir une bonne quantité de restaurants ainsi que des pistes cyclables.

D'un côté nous avons les données concernant le réseau de pistes cyclables de la ville de Québec qui viennent du site https ://www.donneesquebec.ca/fr/ et de l'autre les données sur les restaurants de la ville qui viennent du site https ://www.tripadvisor.fr/

Le dataset des pistes cyclables est disponible sur le site donneesquebec sous format GeoJSON. Afin d'obtenir le dataset des restaurants qui était introuvable sur internet, nous avons scraper le contenu du site web de tripadvisor sur les restaurants de la ville de Québec grâce à un programme que nous avons développé en javascript afin de mettre les données souhaitées dans un document JSON également.

1.2 Exemple de données

Des exemples de donées sont données en annexe 4.1 page 6 et 4.2 page 6

2 Technologies utilisées

2.1 Language de programmation : Javascript

D'un point de vu de la fiabilité, Javascript nous offre tout ce que nous désirons. Nous pouvons utiliser des fonctions tels que des regex pour vérifier les entrées et modifier nos données. Les performances sont bonnes pour notre cas.

Concernant la maintenabilité nous avons de l'expérience en javascript. Nous avons déjà réalisé des projets dans ce language. Nous sommes à l'aise dans l'utilisation de nodejs, d'Express et pourrons donc programmer de manière efficace et permettant le bon fonctionnement du programme. Javascript n'est pas un language très "lourd", il est plutôt facile de comprendre le code. Nous pouvons ajouter des test afin de repérer les problèmes là où cela est nécessaire très rapidement.

Pour ce qui est de l'extensibilité cela nous permet de réaliser tout ce que nous voulons. Il est facile de faire fonctionner les technologies que nous utiliserons soit Docker, NodeJs, Express ou encore Mongoose avec du Javascript. Nous pourrons nous adapter à tout changement dans notre projet, à l'intégration de nouvelles technologies au cours de ce dernier de manière simple, rapide et efficace.

2.2 Bases de données

Nous utilisons MongoDB et Neo4j pour nos bases de données. Plus précisement nous utilisons MongoDB pour la base de données des restaurants car il est alors facile d'utiliser les paires clé-valeurs pour enregistrer les restaurants et faire des requètes selon leur type par exemple ou leur classement. Nous utilisons Neo4j pour les pistes cyclables, nos données correspondent bien à ce genre d'utilisation pour trouver des chemins entre différents lieux, faire des requètes géograhiques et on pourra utiliser Neo4j pour visualiser nos données sous forme de graphe.

Nous utiliserons MongoDB pour sa maintenabilité. Nous avons de l'expérience avec ce type de système de gestion de base de données et pouvons nous assurer d'avoir des requètes rapides qui fonctionnent correctement et qui sont surtout ergonomiques. Neo4j donne de l'extensibilité à notre projet notament par la facilité de réaliser des graphes pour voir les chemins possibles.

3 Processus d'ETL

3.1 Processus d'acquitision initiales des données

Afin d'obtenir les données brutes, il est nécessaire de scraper le site web de *Tripadvisor* (https ://www.tripadvisor.fr/) pour obtenir toutes les données qui nous intéressent sur les restaurants de la ville de Québec. Concernant les données brutes des pistes cyclables elles sont tout simplement téléchargées depuis le site de la ville de Québec (https ://www.donneesquebec.ca/fr/). Des exemples de ces donées sont donnés en annexe 4.3 page 6 et 4.4 page 7

Concernant Tripadvisor, les données qui nous interessent sont dispersées sur les pages tripadvisor de chaque restaurant de la ville.Pour ceci, nous avons utilisé un client nodejs nommé "Axios" qui exécute des requètes get afin de télécharger le code HTML de la page désirée ainsi qu'un module nodejs nommé "Cheerio" qui nous permet de manipuler le code HTML et d'aller récupérer grâce à des selecteurs les données qui nous intéressent.

Avant de récuperer les informations des restaurants nous avons du récuperer leur URL. Nous avons donc scrapé le nombre de pages qui contiennent des liens vers les restaurants de la ville de Québec. Après cela nous parcourons ces pages qui affichent les restaurants de la ville de Québec et scrapons puis stockons dans un tableau chaque URL de restaurants.

Maintenant que nous avons tous les liens vers les pages de nos restaurants nous récuperons les informations que nous souhaitons (voir exemple de données). Pour cela à l'aide de cheerio nous avons selectionnons les informations contenues aux emplacements désirées afin de les stocker dans des variables de type string ou float.

3.2 Processus d'acquisition incrémental de données

Pour l'instant nous n'avons pas implémenté de processus d'aquisition incrémental dans notre projet. En effet lors du lancement de la base de données, toutes les données des restaurants et des pistes cyclables sont supprimées de la BDD MongoDb et Neo4j. Par la suite il sera néanmoins possible d'ajouter, modifier ou supprimer des données. Nous stockerons nos données dans les containers docker en utilisant de la persistance et les fichiers des BDD seront lus à chaque lancement. Les modifications seront rajoutées directement dans le fichier contenant les données.

3.3 Processus de transformation des données transformées

3.3.1 MongoDb et TripAdvisor

Une fois les données stockées dans des variables comme vu précedemment, nous avons du les transformer afin de ne garder seulement les informations qui nous interessent et de normaliser leur affichage.

— name : la donnée scrapé n'a pas du être transformée

— classement : nous avons du effectuer un substring afin de ne garder que les 3 premiers caractères du contenu scrapé avec cheerio afin de ne garder suelement le classement

— address : nous avons utilisé une fonction .first() lors du scraping afin de ne garder que l'adresse contenue dans le selecteur.

— geometry : nous avons du extraire depuis le code source de la page les coordonnées longitude latitude du restaurant qui sont indiquées dans un lien vers Google Map. Pour cela nous avons extrait les coordonnées du code source qui representaient celle du restaurant, transformé le string obtenu en ne gardant que les chiffres, ainsi que '-' et '.' car les coordonnées n'avaient pas tous la même taille. Ensuite nous avons split la longitude et latitude dans un tableau avant de les placer en les dans une variable sous format GeoJSON.

— type : nous avons parcouru les différents élements contenus dans dans un sélecteurs. Nous avons à cet endroit le prix ainsi que les types de cuisine. Nous avons mit tous ces élements dans un tableau puis n'avons gardé dans un deuxième tableau que les types de cuisine.

— price : le prix n'étant pas toujours indiqué pour le restaurant nous vérifions à l'aide d'un regex qu'il est contenu dans le tableau initial regroupant ce dernier et les types de cuisine. S'il n'est pas compris nous mettons de façon arbitraire le prix sous forme de string "NA" (non attribué), et s'il est présent nous retournons dans la variable du prix le string associé du tableau intial.

— link : concernant le lien nous avions stocké les URL de chaque restaurants afin de pouvoir accéder aux données de chacun d'entre eux. Nous rajoutons donc ce dernier également.

Les données sont enreistrées dans un fichier json qui est par la suite lu lors du lancement de la base de donnée MongoDB et inséré dans cette dernière. La base de donnée est située sur un container Docker, et nous nous y connectons en utilisant le driver mongoose.

3.3.2 Neo4j et les pistes cyclables

Comme vu 2.2 page 2 les données des pistes cyclables de la ville de Québec sont stockées dans Neo4j. Les données une fois téléchargées présentent trop de clés. De ce fait seulement trois clés seront utlisées pour définir une piste cyclable :

— ID : Un int qui est le matricule de la piste.
— coordinates : Un tableau deux dimenssion des coordonées qui delimitent la piste.
— LONGUEUR : Un float qui représente la longueur de la piste en mètre.

Ces modifications se font à la volé lors du remplissage de la base de donnnée Neo4j dans le script *populate.js* par le driver Neoj4-driver.

3.4 Schéma de la pipeline d'ETL

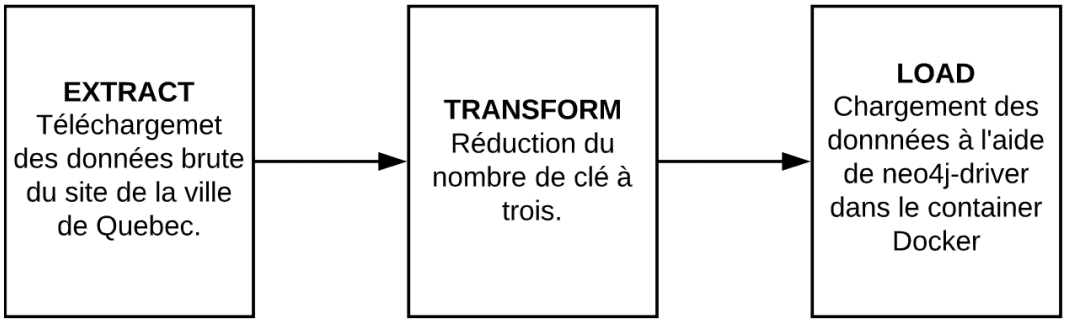

Fig. 1 – Schéma de la pipeline d'ETL pour Neo4j

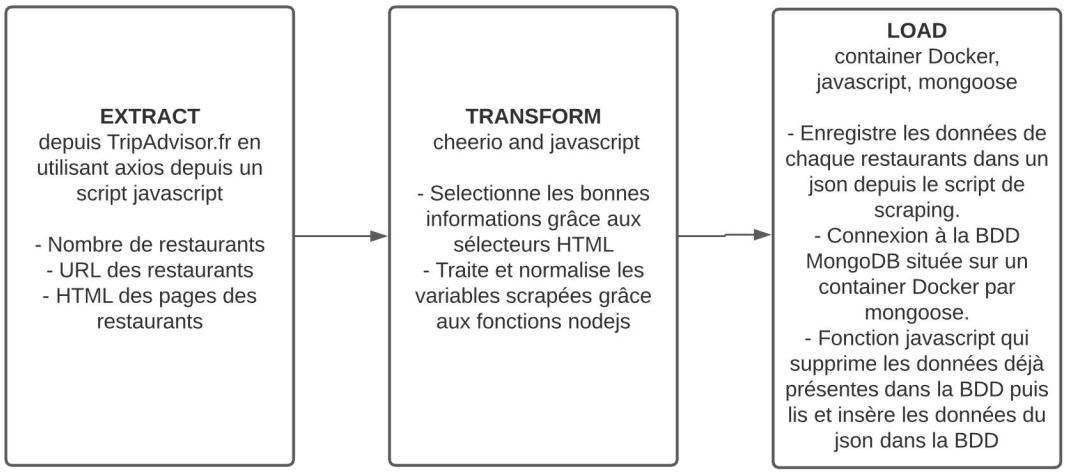

Fig. 2 – Schéma de la pipeline d'ETL pour MongoDb

4 Annexe

4.1 Exemple de données de restaurant


```
{
    ”name” : ”Hotel Manoir Victoria”,
    ”classement” : ”4,5”,
    ”address” : ”44 Cote Du Palais, Québec (ville), Québec G1R 4H8 Canada”,
    ”link” : ”https ://www.tripadvisor.fr/Restaurant_Review-g155033-d979402
            -Reviews-Hotel_Manoir_Victoria-Quebec_City_Quebec.html”,
    ”geometry” : {
        ”type” : ”Point”,
        ”coordinates” : [-71.21089,46.814518]
    }
}
```


4.2 Exemple de données de piste cyclable


```
{
    ”type” : ”Feature”,
    ”geometry” : {
        ”type” : ”LineString”,
        ”coordinates” : [
            [ -71.24206581139462, 46.83160525282929 ],
            [ -71.24284576331979, 46.83131069798517 ]
        ]
    },
    ”properties” : {
        ”ID” : 100096,
        ”NOM_TOPOGRAPHIE” : ”Boulevard des Alliés”,
        ”TYPE” : ”Chaussée désignée”,
        ”SENS_UNIQUE” : ”N”,
        ”LONGUEUR” : 67.92
    }
}
```


4.3 Exemple de données de restaurant après scraping


```
{
    ”name” : String,
    ”classement” : String,
    ”type” : String[],
    ”price” : String,
    ”address” : String,
    ”link” : String,
    ”geometry” : {
        ”type” : String,
        ”coordinates” : float[]
    }
```

4.4 Exemple de données de piste cyclable

```
"type" : String,
"geometry" : {
    "type" : String,
    "coordinates" : float[[]]
},
"properties" : {
    "ID" : int,
    "NOM_TOPOGRAPHIE" : String,
    "TYPE" : String,
    "SENS_UNIQUE" : String,
    "LONGUEUR" : float
}
```

5 Remarques

Le programe est un peu long au démarage notament le lancement de neo4j et l'importation des données dans la BDD, par conséquent les routes GET/ peuvent ne pas fonctionner dans les premieres dizaines de secondes après un $docker-compose up. De plus une landing page est disponible à l'adresse '`localhost:80/`' elle permet d'accéder au differentes routes GET/ plus facilement. Un message indiquant la bonne importation des données dans Neo4j puis dans MongoDB s'affiche dans le terminal de lancement et induit le fait que les requètes sont dès lors possibles.