Université Laval

Bases de données avancées

GLO-7035/4035

Projet longitudinal

Dérisquer l'application

Billot Mathias NI : 536 927 197
Labrie Martin NI : 996 106 840
Vandermeersch Thomas NI : 537 025 581

Automne 2022

Table des matières

I. Stratégie d'acquisition des données

a) Source et méthode d'extraction

Pour ce travail, nous avons décidé de travailler sur la ville de New-York (États-Unis). Pour ce faire, nous avons récupéré deux jeux de données sur le site officiel de la ville de New-York concernant les données ouvertes [1].

Le premier jeu de données utilisé contient l'ensemble des pistes cyclables de la ville de New-York [2]. Ce jeu de données est téléchargeable directement au format GeoJSON et est mis à jour annuellement par le *Department of Transportation* (Ministère des Transports) de New-York, ce qui lui donne une grande fiabilité.

Le second jeu de données contient la liste des restaurants inspectés par le *Department of Health and Mental Hygiene* (Ministère de la santé et de l'hygiène mentale) de New-York [3]. Ce département supervise l'inspection de tous les restaurants de New-York et fournit, selon des critères d'inspection publics, un score sur la compétence de l'établissement à respecter les règles d'hygiène en place. Ce jeu de données nous fournit donc des données très pertinentes, car en plus d'avoir la liste des restaurants et leur localisation, nous avons des informations additionnelles telles que leur nom, leur adresse, leur quartier, le type de cuisine qu'il propose, et un score de qualité. De plus, les données sont mises à jour automatiquement tous les jours.

Ces deux jeux de données nous semblent donc très pertinents, car ils sont fiables et riches en information. Pour l'instant, les données sont téléchargées et introduites dans le projet manuellement. Pour la suite, elles seront automatiquement téléchargées par un script récurrent sur la base de la fréquence de mise à jour des données par les émetteurs.

b) Présentation d'exemples de données sources

Le jeu de données sur les pistes cyclables contient plus de 20 800 entrées au format GeoJSON. Chaque entrée contient des informations additionnelles telles que le nombre de voies cyclables, les rues concernées, le quartier et une localisation précise. Retrouvez un exemple d'entrée à l'annexe "Extrait du jeu de données des pistes cyclables".

II. Technologies utilisées

a) API REST : *Node.js* & *Express*

Nous avons décidé d'utiliser *Node.js* [4] comme langage de programmation et nous utiliserons la librairie *Express* [5] pour créer l'API REST et le serveur web.
Node.js a les avantages d'être :

1. NYC Open Data : https://opendata.cityofnewyork.us/
2. Jeu de données des pistes cyclables : https://data.cityofnewyork.us/Transportation/New-York-City-Bike-Routes/7vsa-caz7
3. Jeu de données des restaurants : https://data.cityofnewyork.us/Health/DOHMH-New-York-City-Restaurant-Inspection-Results/43nn-pn8j
4. Node.js : https://nodejs.org/fr/
5. Express : https://expressjs.com/fr/

— Fiable : Le code de *Node.js* est asynchrone (et donc non bloquant). Aussi, la grande communauté autour de *Node.js* et *Express* leur procure une grande fiabilité.
— Extensible : *Node.js* vient avec le gestionnaire de dépendances *npm*. Cette bibliothèque gigantesque contient des modules pour tous les cas d'utilisation possible.
— Maintenable : Le module *Express* est téléchargé 25 880 051 de fois par semaine [6] et le module *mongoose* (connexion à une base de données *MongoDB*) 1 970 914 par semaine [7]. Ils sont donc constamment maintenus.

b) Bases de données : *MongoDB* & *Neo4j*

MongoDB est une base de données orientée documents. Nous utiliserons cette base de données pour stocker les différents restaurants et les pistes cyclables. *MongoDB* est intéressant pour la manipulation de données géographiques (Trouver les restaurants à proximité).

D'autres avantages de *MongoDB* :

— Extensibilité (1) : Comme le schéma n'est pas défini à l'avance, il est possible d'ajouter très facilement de nouveaux champs aux collections en cours de projet.
— Extensibilité (2) : Si le nombre d'utilisateurs ou de données de notre application devenait trop important, *MongoDB* permet des solutions de *clustering* robustes.
— Maintenabilité et fiabilité : *MongoDB* est la base de données *NoSQL* la plus populaire en 2022 [8].

Neo4J est une base de données orientée graphe. Ce type de base de données est optimisé pour connecter des données. Nous aurons donc des points reliés par des routes. Nous pourrons ensuite demander à *Neo4j* de calculer les points qu'il faut traverser pour aller d'un point A à un point B, et ce, avec des notions de poids sur les routes (qui seraient le nombre de km).

D'autres avantages de Neo4J

— Extensibilité (1) : Comme le schéma n'est pas défini à l'avance, il est possible d'ajouter très facilement de nouveaux champs aux nœuds et de nouvelles relations.
— Extensibilité (2) : Si le nombre d'utilisateurs ou de données de notre application devenait trop important, *Neo4j* permet des solutions de *clustering* robustes.
— Fiabilité et maintenabilité : *Neo4j* est la 5ème base de données NoSQL la plus populaire en 2022 [9]. Au vu du nombre d'utilisateurs, elle se doit d'être fiable et maintenue. Des nouvelles versions sont régulièrement publiées.

c) Architecture du projet

Afin de faciliter la compréhension du projet, voici l'explication de l'architecture du projet remis et de ses fichiers.

— `/back-api`
Contient le projet Node.js, le serveur web

6. Statistiques d'*Express* : https://www.npmjs.com/package/express
7. Statistiques de *mongoose* : https://www.npmjs.com/package/mongoose
8. Popularité de *MongoDB* : https://db-engines.com/en/ranking
9. https://www.decipherzone.com/blog-detail/nosql-databases

— `/datasets`
Contient les fichiers de données originaux et transformés. Ce dossier est populé par le script ETL
— `/db-mongodb`
Contient les fichiers du container MongoDB
— `/db-neo4j`
Contient les fichiers du container Neo4j
— `/script-etl`
Contient le script Python ETL
— `/.env`
Fichier contenant les variables d'environnement partagées par les containers et l'app web
— `docker-compose.yml`
Fichier contenant la configuration des trois containers Docker
— `README.md`
Fichier contenant la présenation du projet

III. Détails du processus d'extraction, de transformation et de conversion (ETL)

Afin d'effectuer le processus continu d'extraction, de transformation et de conversion des données, nous avons créé un script Python. Ce script est un outil configurable via les paramètres afin de réaliser le processus partiellement ou en totalité. La documentation de ce script peut être trouvée à l'annexe "Documentation du script ETL Python".

a) Processus d'acquisition initiale des données

La première étape du processus ETL est l'extraction. Cette étape est relativement simple car elle consiste à télécharger le fichier de données afin de l'utiliser dans les étapes suivantes. Pour cela, Python dispose d'un outil `urlretrieve` qui permet de télécharger un fichier et de le placer sur le disque. Dans notre cas, les données sont placées dans le dossier `/datasets` du projet.

b) Processus d'acquisition incrémental des données

Afin de ne pas avoir à recalculer et réinsérer toutes les données à chaque mise à jour du jeu de données original, deux techniques sont utilisées :

1. Pour les restaurants, avant d'insérer chaque donnée, le script vérifie si l'identifiant unique du restaurant n'existe pas déjà dans la base de données.
2. Pour les pistes cyclables, la concaténation de leurs attributs permet de générer un hash unique qui identifie la piste. Si un attribut venait à changer ou si une nouvelle piste était insérée, alors uniquement celles-ci seraient ajoutées dans la base de données.

Ce processus devrait être exécuté de manière récurrente dans un environnement de production. Dans notre cas, il reste manuel à des fins de démonstration.

c) Processus de transformation des données

Cette étape consiste à traiter les données afin de pouvoir par la suite les intégrer dans les deux bases de données.

Pour cela, trois étapes sont réalisées :

1. Transformer les données originales pour ne garder que les informations nécessaires et les formater :

   Pour les pistes cyclables, cela consiste à garder quatre informations : le nom de la rue de la piste cyclable, le quartier dans lequel elle se trouve, la direction de la piste cyclable (sens unique ou double voie) et enfin la position géospatiale de la piste.
   On calcule aussi un hash unique qui représente la piste cyclable en hachant toutes les données précédentes. Cette opération servira tard à ne pas devoir réinsérer toutes les données, mais uniquement celles qui ne se trouvent pas encore dans la base de données.
   Pour les restaurants, cela consiste à ne garder d'un enregistrement par restaurant (car le jeu de données contient plusieurs évaluations de l'agence de la santé par restaurants), extraire les informations intéressantes (voir annexe), et filtrer les données qui ne contiennent pas une des informations requise telle qu'un nom, une position, une note satisfaisante, etc.

2. Trouver pour chaque restaurant la piste cyclable la plus proche :

   Cette étape est cruciale afin de générer notre graphe. Elle permet de trouver pour chaque restaurant, la piste cyclable la plus proche dans un rayon de 100 mètres. Si une telle piste cyclable n'existe pas, le restaurant est retiré de notre jeu de données. Cette opération est **très** couteuse en calcul. En effet, la partie du script prend en moyenne **7 à 8 heures** à se terminer du fait de la complexité de trouver le point le plus proche dans un échantillon de 21.000 pistes cyclables. C'est pour cette raison qu'il est possible de passer cette étape si la carte d'association est disponible et déjà calculée.

3. Formater les données dans des formats prêts à être intégrés dans les bases de données :

   Enfin, la dernière étape récupère les données formatées (les restaurants, les pistes cyclables et la carte d'association de la piste cyclable la plus proche pour chaque restaurant) et les transforme en données prêtes à être insérées dans les bases de données. Il en resort alors deux fichiers : un fichier `JSON` à insérer dans la base de données MongoDB et un fichier `Cypher` qui contient la liste des instructions pour ajouter les pistes cyclables et les restaurants dans la base de données Neo4j.

d) Schéma de la pipeline d'ETL

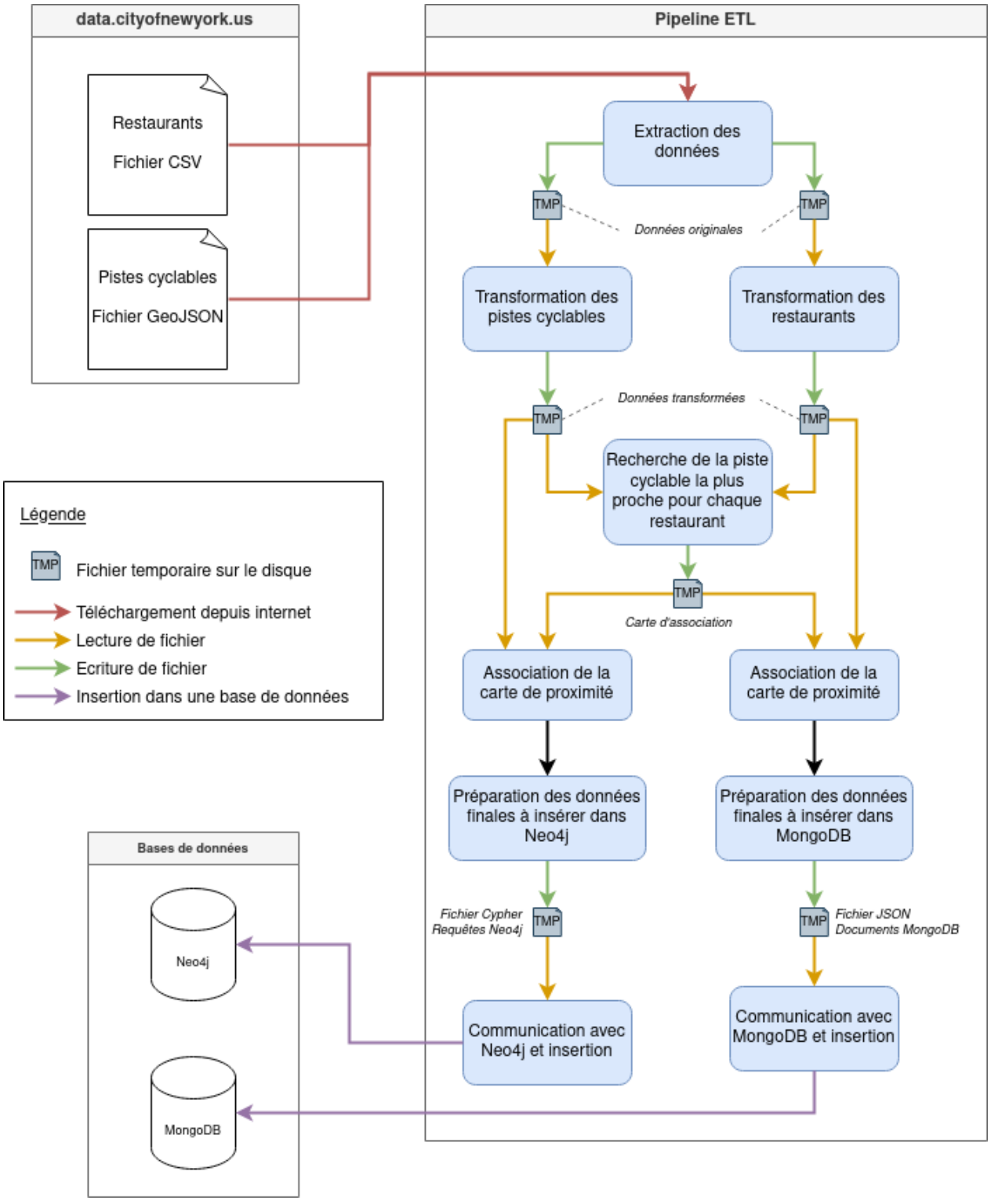

FIGURE 1 – Pipeline ETL du projet

IV. Annexes

a) Extrait du jeu de données des pistes cyclables

```
{
    "type": "FeatureCollection",
    "features": [
        ...
        {
            "type": "Feature",
            "properties": {
                "bikedir": "2",
                "fromstreet": "BATTERY PARK GREENWAY",
                "lanecount": "2",
                "facilitycl": "I",
                "ft2facilit": null,
                "tf2facilit": null,
                "tostreet": "W 59 ST",
                "tf_facilit": "Greenway",
                "allclasses": "I",
                "onoffst": "OFF",
                "ft_facilit": "Greenway",
                "boro": "1",
                "street": "HUDSON RIVER GREENWAY",
                "shape_le_1": "493.975247086",
                "segmentid": "260472",
                "comments": null
            },
            "geometry": {
                "type": "MultiLineString",
                "coordinates": [...]
            }
        }
        ...
    ]
}
```

Extrait de code 1 – Extrait du jeu de données des pistes cyclables

b) Extrait du jeu de données des restaurants

```
{
    "type": "FeatureCollection",
    "features": [
        ...
        {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -73.987301,
                    40.722504
```

```
                ]
            },
            "properties": {
                "CAMIS": 40732665,
                "DBA": "KATZ'S DELICATESSEN",
                "BORO": "Manhattan",
                "BUILDING": "205",
                "STREET": "EAST HOUSTON STREET",
                "ZIPCODE": 10002,
                "PHONE": "2122542246",
                "CUISINE DESCRIPTION": "Sandwiches",
                "SCORE": 29,
                "GRADE": null,
                "Latitude": 40.7225041175,
                "Longitude": -73.9873009612
            }
        }
        ...
    ]
}
```

Extrait de code 2 – Extrait du jeu de données des restaurants

c) Documentation du script ETL Python

```
$> python3 script-etl/main.py -h
usage: main.py [-h] [-e] [-t] [-l] [--load-mongodb] [--load-neo4j] [--truncate] [--
    debug] [--symlines] [--skip-nearest-map]

Run ETL script pipeline for GLO-7035 application

options:
  -h, --help           show this help message and exit
  -e, --extract        Extract data from source and get raw datasets
  -t, --transform      Transform both raw datasets
  -l, --load           Load read-to-import datasets into both mongodb and neo4j
  --load-mongodb       Load read-to-import datasets into mongodb only
  --load-neo4j         Load read-to-import datasets into neo4j only
  --truncate           Dump databases before insertion
  --debug              Display some debug information
  --symlines           Enable symbolic lines creation on transformation step
  --skip-nearest-map   Skip nearest map step (may take up to 7 hours)
```

Extrait de code 3 – Documentation de l'outil

d) Carte d'association

```
...
50009734,711c95d9c619575c1668
41378724,8df1e99965fabcbda694
50000577,f9cbabb73e9790d10905
```

```
41379121,eda3e870f58f6e483f97
41380930,6edd30b2e3cbee19bd6f
...
```

Extrait de code 4 – Extrait de la carte d'association entre les restaurants et les pistes cyclables. Fichier CSV

e) Données Cypher

```
...
MERGE (r:Restaurant { uid: 50058097, name: "CROWN FRIED CHICKEN", grade: "A", cuisineType: "Chicken" })
MERGE (r:Restaurant { uid: 50058113, name: "4 CHARLES RIB PRIME RIB", grade: "A", cuisineType: "Steakhouse" })
MERGE (r:Restaurant { uid: 50058121, name: "MOE'S SOUTHWEST GRILL", grade: "A", cuisineType: "Mexican" })
MERGE (r:Restaurant { uid: 50058123, name: "KITH TREATS", grade: "A", cuisineType: "Frozen Desserts" })
...
MERGE (i:Intersection { hash: "77b6844806463af0060b" })
MATCH (from:Intersection { hash: "dca848c4bf62806cf410" }) MATCH (to:Intersection { hash: "77b6844806463af0060b" }) MERGE (from)-[:goes_to { uid: "7f95cc2a2e50a90d93a4", distance: 14.0 }]->(to)
MERGE (i:Intersection { hash: "8d79cbf9c552c231eb29" })
MATCH (from:Intersection { hash: "77b6844806463af0060b" }) MATCH (to:Intersection { hash: "8d79cbf9c552c231eb29" }) MERGE (from)-[:goes_to { uid: "7f95cc2a2e50a90d93a4", distance: 14.0 }]->(to)
MERGE (i:Intersection { hash: "c48d99badb3cd49a7d66" })
...
MATCH (r:Restaurant { uid: 40928958 }) MATCH (i:Intersection)<-[:goes_to { uid: "bbc2acc3024d377aac0b" }]-(:Intersection) MERGE (r)-[:is_at]->(i)
MATCH (r:Restaurant { uid: 50093628 }) MATCH (i:Intersection)<-[:goes_to { uid: "822cf01a414b0f30423e" }]-(:Intersection) MERGE (r)-[:is_at]->(i)
MATCH (r:Restaurant { uid: 40929097 }) MATCH (i:Intersection)<-[:goes_to { uid: "bb9f71a013f4d71dd4c2" }]-(:Intersection) MERGE (r)-[:is_at]->(i)
MATCH (r:Restaurant { uid: 41418714 }) MATCH (i:Intersection)<-[:goes_to { uid: "423d6be1f513db506926" }]-(:Intersection) MERGE (r)-[:is_at]->(i)
MATCH (r:Restaurant { uid: 50093702 }) MATCH (i:Intersection)<-[:goes_to { uid: "a168bbe791d2541a36ef" }]-(:Intersection) MERGE (r)-[:is_at]->(i)
...
```

Extrait de code 5 – Extrait du fichier Cypher contenant les requêtes Neo4j à insérer dans la base de données