GLO-7035 : Bases de données avancées

Travail longitudinal

UNIVERSITÉ LAVAL

Réalisé par :

Nathan Krulic

Faustin Maniragena

Session automne 2022

I/ Introduction

1. Explication de la problématique

Le but de ce projet est de proposer une application qui a pour but de tracer un trajet en vélo dans une ville donnée, en passant par plusieurs restaurants. L'utilisateur doit pouvoir renseigner un nombre d'arrêts souhaité, la distance à parcourir et sélectionner des types de restaurants parmi une liste.

2. Sommaire de la solution

Pour notre solution, nous allons chercher les données de restaurants et de pistes cyclables sur le site du gouvernement du Québec, qui comporte des jeux de données pour plusieurs domaines : tourisme, transport, évènements etc ... Nous sauvegardons ces données en base de données afin de pouvoir les traiter par la suite. Avec ces données, le but va être de construire une base de données en graphe, représentant la connexion entre les différentes pistes cyclables et les restaurants. Ainsi, un nœud sera un restaurant, et les arêtes/liaisons seront les pistes cyclables. Cela permettra une meilleure visibilité et un temps de traitement bien plus rapide une fois le graphe créé.

3. Présentation du rapport

Dans ce rapport, nous allons d'abord vous présenter la stratégie d'acquisition des données avec les sources et méthodes d'extraction et la présentation des données sources. En 3e partie nous verrons les technologies utilisées : langage de programmation et bases de données. Ensuite en 4ème partie nous allons voir le détails du processus d'extraction, de transformation et de conversion (ETL). Puis en 5ème partie, nous allons voir les détails du pipeline de données en présentant l'algorithme permettant de produire les parcours et l'algorithme de calcul permettant de trouver le parcours épicurien le plus intéressant. Enfin, en 6 nous présenterons une explication du plan d'expansion de l'application. Pour finir, en dernière partie nous présenterons les fonctionnalités additionnelles avancées.

II/ Stratégie d'acquisition de données

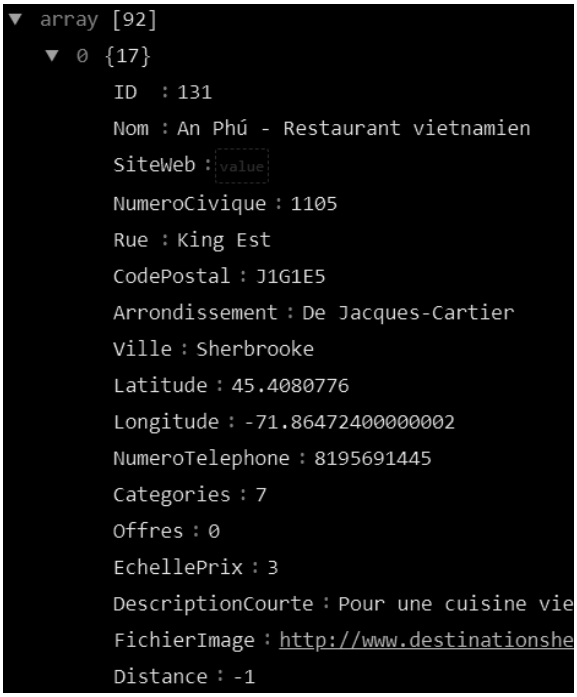
1. Source et méthode d'extraction

Comme source de données, nous avons choisi d'utiliser le portail de données Québec (https://www.donneesquebec.ca/), qui contient une multitude de jeux de données dont des restaurants et pistes cyclables. Pour convenir à notre application nous avons dû chercher 2 jeux de données dans la même ville, et donc nous avons choisi Sherbrooke.

La méthode d'extraction pour le moment est très primitive et voué à évoluer. En effet, nous nous contentons de télécharger les fichiers JSON des jeux de données, pour ensuite les parser et tout ajouter à la base de données, dans les collections respectives. Cette méthode à pour inconvénient que la base de données est en fichier brut .json dans l'application donc cela la rend plus lourde, et aussi, si le jeu de donnée est mis à jour avec de nouveaux restaurant par exemple, cela ne sera pas pris en compte. C'est actuellement une solution temporaire, et nous prévoyons de lire le fichier directement depuis son URL dans notre application. Ce rapport sera donc mis à jour en conséquence.

2. Présentation d'exemples de données sources

Comme illustration, voici un exemple d'un restaurant modélisé dans notre jeu de donnée dans le fichier .json :

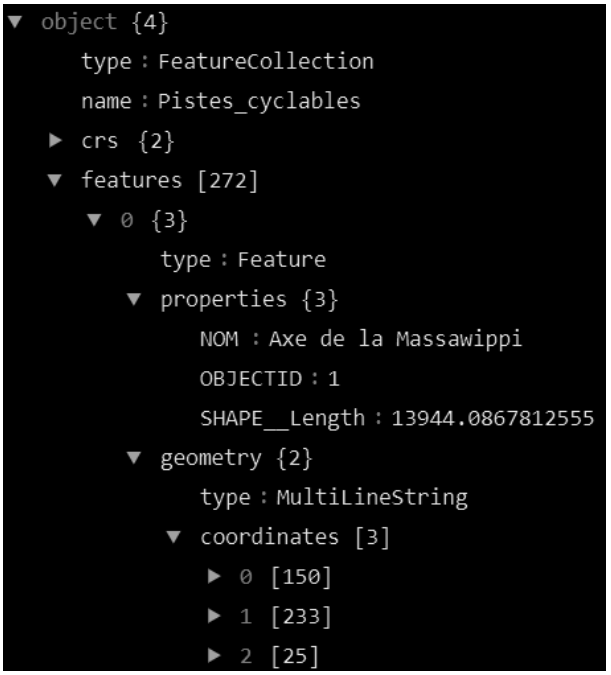


Le fichier contient donc un tableau ici de taille 92, rempli de restaurants.

On peut y retrouver les informations les plus importantes tel que le nom, un ID de catégorie qui correspondent chacun à un type de restaurant (fast-food, Asiatique, Pizza etc ...), l'adresse et les coordonnées GPS qui vont nous permettre de situer le restaurant et calculer des distances avec nos pistes cyclables.

Aussi, il est à noter qu'un restaurant peut avoir plusieurs catégories.

Enfin, voici un exemple de piste cyclable de notre jeu de données :



Cette fois-ci, le json contient un objet avec 4 attributs : un type, un nom, "crs" et un tableau features.
Nous ne nous intéressons qu'à l'attribut features car il contient toutes les pistes cyclables.
Ici nous avons développé une piste cyclable. On peut voir qu'un objet dans le tableau features possède 3 attributs : un type, des propriétés dans lesquelles on retrouve le nom de la piste cyclable, un ID, et une longueur de piste cyclable, et enfin un attribut geometry qui contient en fait un tableau de coordonnées qui servent à dessiner la piste cyclable. Ainsi, dans notre exemple, on a 3 lignes qui contiennent chacune respectivement 150, 233 et 25 coordonnées GPS que nous n'avons pas développé ici pour des soucis de lisibilité. Encore une fois, grâce à ces coordonnées nous allons pouvoir faire des calculs de distance.

II/ Technologies utilisées

1. Langage de programmation

Nous avons choisi de développer notre application Java en utilisant le framework Spring. Cela nous permet de développer une API qui est facilement extensible pour le futur de l'application, et nous permet de construire une couche serveur complètement indépendante afin d'y ajouter plus tard une interface graphique par le biais d'une application Web par exemple. Spring est également connu pour sa rapidité d'exécution et de shutdown.

2. Base de données

Pour ce qui est des bases de données, nous avons choisi d'utiliser MongoDB pour le stockage des données de restaurants et de pistes cyclables dans 2 collections distinctes et Neo4J pour notre graphe qui fait la liaison entre ces 2 entités.

En utilisant MongoDB, la transition Json > Java Spring > Collection MongoDB peut se faire simplement grâce à la syntaxe Json-like des collections mongoDB. On profite également du modèle en document qui est une manière efficace de stocker et récupérer des données et avec rapidité.
Dans notre base de données mongo, nous avons 3 collections : Bikeroute, qui correspond aux pistes cyclables, Restaurants pour la liste des restaurants, et enfin TypeRestaurant qui répertorie les types de restaurants et est en liaison avec Restaurants.

Neo4J grâce à sa structure en graphe va nous donner des hautes performances pour nos requêtes de traçage de chemin. Aussi, on peut voir le graphe comme une visualisation "naturelle" des chemins entre les restaurants, ce qui rend l'utilisation de la base de données plus intuitive.
Pour le moment, notre base de données Neo4J est vide car nous n'avons pas encore écrit l'algorithme de remplissage.

IV/ Les détails du processus d'extraction, de transformation et de conversion (ETL)

1. Processus d'acquisition initiale des données

Le processus d'acquisition initiale des données se fait pour le moment avec un fichier .json à la racine du projet de l'application spring. Nous lisons ce fichier (un fichier pour chaque collection) pour pouvoir en extraire les objets Java associés et les ajouter en BD.
La lecture se fait grâce à la bibliothèque "jackson" qui définit une classe ObjectMapper qui permet de mapper une chaîne de caractère sous forme JSON, à un objet associé Java avec les mêmes attributs. Pour nous, la chaîne de caractère est extraite du fichier. L'ObjectMapper

va créer des classes DTO (Data transfer Object) qui représentent en fait les objets “bruts” tel qu’ils sont dans le fichier JSON

```
public class RestaurantDTO {
    1 usage
    @JsonProperty("ID")
    private Double id;
    2 usages
    @JsonProperty("Nom")
    private String nom;
```

Ici par exemple nous avons la classe RestaurantDTO avec les attributs id et nom. L’annotation @JsonProperty nous permet de mapper le champ “ID” du fichier Json avec l’attribut id de la classe DTO.

Le DTO est un pattern d’architecture souvent utilisé en architecture logicielle. Il permet d’ajouter une couche entre une API et un client. De ce fait, en définissant un mapper par la suite, cela permet d’ajouter ou de cacher des champs, comme un mot de passe lorsqu’un utilisateur veut se connecter par exemple. Dans notre cas, nous nous en sommes servi pour le processus de transformation des données.

2. Processus d’acquisition incrémental des données

Actuellement, notre application ne contient pas de processus d’acquisition incrémental des données, mais nous travaillons dessus.
Nous prévoyons par la suite, de comparer nos données en base (mongoDB) avec celles fournies par l’URL.
Actuellement, à chaque démarrage nous regardons seulement si les collections ne sont pas vie. Une solution pour l’acquisition incrémental serait de mettre en place un scheduler chargé de vérifier constamment les données. Cette méthode est bien adaptée à un projet Spring puisque notre application est un serveur censé tourné constamment. Le principe du scheduler est de prévoir de lancer une méthode à un moment donné. Pour l’appliquer, on utilise l’annotation @Scheduled sur une méthode, accompagnée en paramètre d’une expression *Cron*. Cette expression Cron représente en fait l’intervalle à laquelle notre méthode va être appelée.

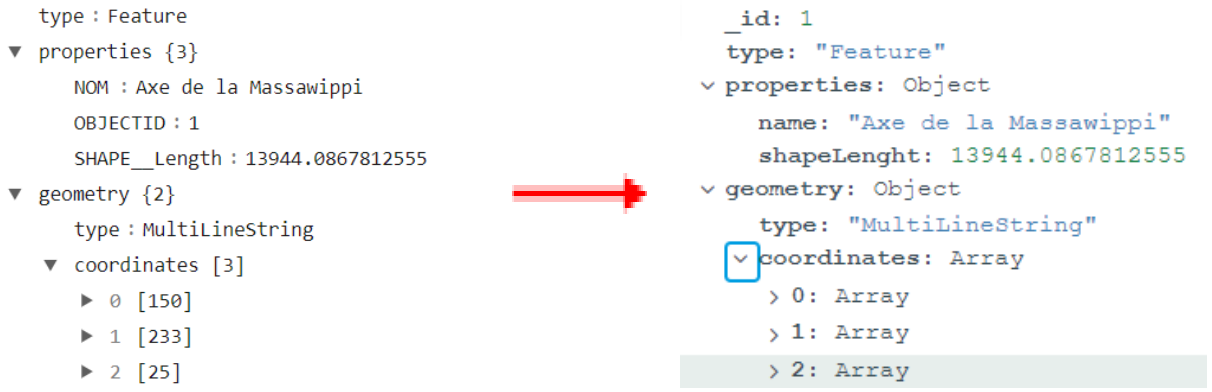
```
@Scheduled(cron = "0 15 10 15 * ?")
public void scheduleTaskUsingCronExpression() {
```

L’expression Cron ci-dessus permet de lancer la méthode à 10:15AM le 15 de chaque mois.

Les champs sont dans l'ordre suivant : secondes, minutes, heure, jour du mois, mois, jour de la semaine. Le symbole * sert à indiquer n'importe quelle valeur, et pareil pour le ? qui est spécifique au jour de la semaine.
Une telle méthode Scheduled pourrait donc être ajoutée pour chacun de nos Services Spring : Restaurant, Bikeroute, et TypeRestaurant. Dans cette méthode, que l'on pourrait exécuter par exemple une fois par mois qui serait une fréquence de mise à jour raisonnable, on va contacter donneesquebec.ca pour aller chercher les données JSON correspondant. Comme les ID des objets dans notre base de donnée mongoDB sont les mêmes que sur donneesquebec.ca, on peut vérifier facilement si des nouvelles données sont à ajouter en faisant un parcours de ces derniers et en comparant le champ ID, avec nos documents sur mongoDB.
Pour le prochain rendu, nous travaillerons sur cette implémentation, en réglant peut être le Scheduler pour qu'il s'exécute toutes les minutes simplement pour des fins d'illustration.

3. Processus de transformation des données

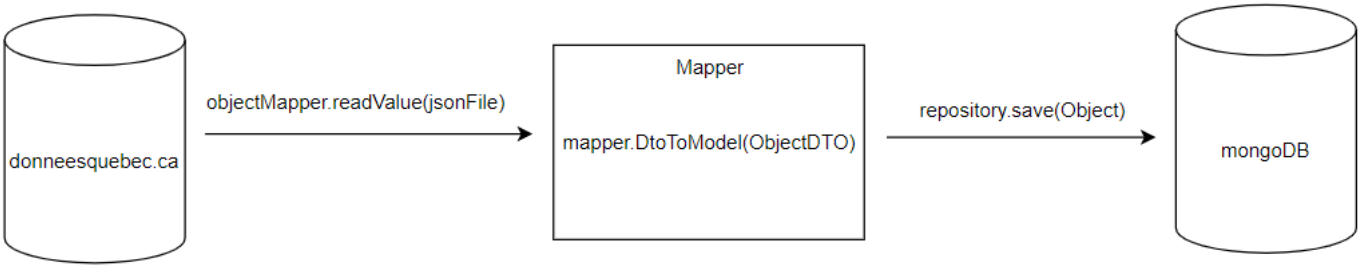
Comme dit précédemment, dans le processus d'acquisition des données, nous utilisons un pattern DTO pour l'acheminement des données vers notre BD mongoDB. De ce fait, on récupère d'abord les données brutes à travers une classe DTO et des annotations de la bibliothèque Jackson comme @JsonProperty, puis grâce à un mapper, on convertit ces DTO en classes du modèle.
Le mapper s'occupe de la transformation des données. Il est instancié dans les méthodes de chargement dans les Services Spring. Grâce à ce mapper, on peut personnaliser les objets à notre guise afin de les remodeler. Par exemple, dans l'objet Restaurant certains attributs comme la ville (puisque nous nous occupons seulement de Sherbrooke actuellement) ne nous sont pas utiles, donc on ne les garde pas pour la classe modèle. Pour l'objet Bikeroute (les pistes cyclables), nous utilisons le mapper pour récupérer l'ID qui, de base, se situe dans un sous-attribut de l'objet JSON, et l'utiliser comme ID mongoDB. Nous avons un mapper pour chaque objet récupéré sur le site donneesquebec.ca : RestaurantMapper, TypeRestaurantMapper et BikerouteMapper, qui sont utilisés dans leurs services respectifs.



Ci-dessus, un exemple de transformation d'une donnée de piste cyclable, vers un document dans notre base de données MongoDB.

Cette transformation permettra la manipulation des données dans notre application, en prévention des futurs calculs de trajectoire, et ainsi la création du graphe sur Neo4J.

4. Schéma de la pipeline d'ETL



Ce modèle de pipeline s'applique donc pour chacun de nos objets/documents mongoDB.
On a à chaque fois un Mapper et un repository associé.

Annexes

Modélisation UML :