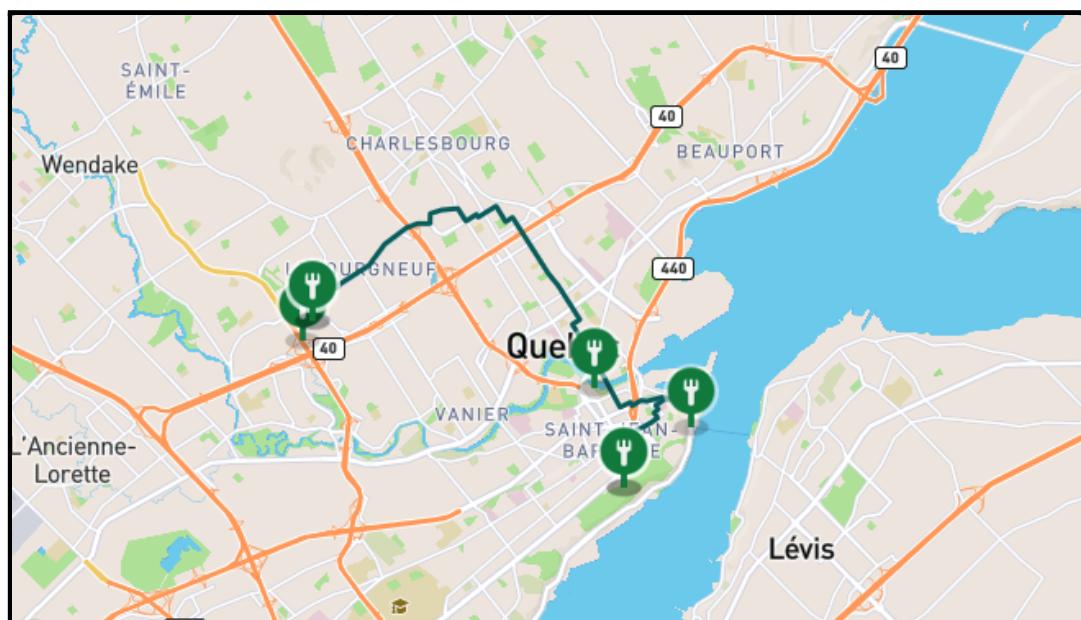


GLO-7035 : Bases de données avancée

Remise 3 : Présentation du MVP



Sommaire

Introduction	3
1. Stratégie d'acquisition des données	3
1.1 Source des données utilisées	3
1.2 Présentation des données	3
2. Technologies utilisées	4
2.1. Langage de programmation utilisé	4
2.2. Technologies de persistance	4
2.2.1. MongoDB	4
2.2.3. Neo4J	4
3. Détails du processus ETL	5
3.1. Obtention des données initiales et incrémentales	5
3.2. Transformation des données	6
4. Détails de la pipeline des données	7
4.1 Création d'un parcours	7
4.2 Obtention d'un parcours	8
5. Explication du plan d'expansion	9
5.1. Gestion d'un volume plus grand de données et d'utilisateurs	9
5.1.1. Stratégie de réPLICATION	9
5.1.2. Stratégie de partitionnement	9
5.1.3. Sécurité des bases de données	10
5.2. Fonctionnalités additionnelles pouvant être implémentées grâce à la structure des données	10
6. Fonctionnalités avancées	11
7. Annexes	12
7.1. Exemples de données	12
7.1.1. Données de restaurants	12
7.1.2. Données de routes	13
7.2. Exemples de transformation des données	13
7.2.1. Transformation des données de restaurants	13
7.2.2. Transformation des données de pistes cyclables	14
Bibliographie	16

Introduction

Nous cherchons à créer une application permettant à un utilisateur d'obtenir un parcours à vélo sur lequel sont situés les restaurants d'une région, plus précisément la ville de Québec. Le public cible de cette application est pour le moment les habitants de la ville de Québec mais pourra être étendu par la suite aux habitants des villes les plus peuplées de la planète. La première étape de la mise en place de cette application consiste à acquérir des données de restaurants et des pistes cyclables situés à Québec. Nous devons alors choisir des technologies adéquates pour récupérer ces données dans le format adapté au cas d'utilisation. C'est ce que nous avançons dans la deuxième section. Quant à la troisième section, elle présente les différentes étapes nécessaires afin d'avoir les données prêtes pour l'utilisation de l'application. Dans la quatrième section, nous rentrerons davantage dans les détails de notre pipeline de données. Plus spécifiquement, nous parlerons du travail effectué sur le parcours, de sa création à son obtention. Finalement, dans la dernière section, nous présenterons notre stratégie d'expansion.

1. Stratégie d'acquisition des données

1.1 Source des données utilisées

Pour ce projet, nous utiliserons des données provenant de OpenStreetMap, que ce soit pour les données des restaurants comme pour les données des pistes cyclables. OpenStreetMap est une base de données géographiques qui couvre le monde entier et qui est entièrement Open-Source.

Pour récupérer les données d'OpenStreetMap nous utiliserons l'API Overpass, qui est une API publique permettant de récupérer les données d'OpenStreetMap.

1.2 Présentation des données

Nous allons donc récupérer la localisation de restaurants, fast-foods ou cafés situés dans la ville de Québec à l'aide de cette API. L'utilisateur envoie une requête GET à l'API pour récupérer l'ensemble des données qui correspond à la requête. Il est possible de récupérer des données au format JSON ou au format GeoJSON. Ainsi, nous pouvons récupérer 1 688 restaurants, fast-foods ou cafés avec la requête située en annexe (figure 1) pour la ville de Québec. De plus, dans OpenStreetMap nous pouvons récupérer des informations sur la cuisine des restaurants comme par exemple "burger", "pizza" ou "sushi".

Par ailleurs, pour les routes cyclables à Québec nous utilisons le même procédé de récupération des données grâce à l'API Overpass. Nous récupérons ainsi les données au format GeoJSON (figure 6) à l'aide d'une seconde requête (figure 4) qui nous permet d'extraire 2417 chemins composés de 23119 positions géographiques.

2. Technologies utilisées

2.1. Langage de programmation utilisé

Nous choisissons d'utiliser pour ce projet le langage Javascript avec un serveur Web Node JS. Ce choix a été fait pour plusieurs raisons. Nous avons choisi Node Js car c'est un environnement Opensource d'exécution fiable et plébiscité par la communauté de développeurs web. Node Js respecte les critères d'extensibilité car il permet de gérer un grand nombre de connexions simultanément avec une bonne performance et permet facilement l'évolution des applications. De plus, la possibilité d'y importer des packages open source via npm ajoute de la flexibilité à notre application. Nous pourrons ainsi, si nous le souhaitons, ajouter une fonctionnalité en important un package npm nous permettant d'effectuer la tâche voulue.

Le critère de maintenabilité est aussi respecté : la facilité de programmation d'un serveur Node JS est parfaitement adaptée pour notre usage. Nous pouvons ainsi utiliser le même langage de programmation pour créer le front-end et le back-end.

2.2. Technologies de persistance

2.2.1. MongoDB

Tout d'abord, la première technologie de persistance que nous utiliserons sera une base de données MongoDB. Nous utiliserons cette base de données pour stocker les données relatives aux restaurants.

En tant que base de données orientée documents, MongoDB permet aux développeurs de stocker facilement des données structurées ou non structurées. Elle utilise un format de type JSON pour stocker des documents. Ce format correspond directement aux objets natifs dans la plupart des langages de programmation modernes, ce qui en fait un choix naturel. Il nous permet de ne pas normaliser les données. Cela correspond parfaitement à notre besoin car les données des restaurants peuvent avoir des formes différentes, avec des champs présents ou non. Nous pourrons ainsi avoir un schéma de données flexible et capable de gérer facilement une recherche par catégorie ou type de nourriture. MongoDB peut également gérer un volume élevé et peut évoluer pour s'adapter à de grandes charges de données. MongoDB est très adapté aux données sous format JSON ce qui correspond parfaitement aux données que nous possédons.

2.2.3. Neo4J

Nous utiliserons Neo4j, une base orientée graphes, comme seconde technologie de persistance, pour les pistes cyclables. Neo4j permet de représenter les données en tant que nœuds, reliés par un ensemble d'arêtes. Les nœuds servent à représenter des entités (comme des objets physiques, des personnes ou des concepts) et contiennent des informations sous forme de propriétés. Les nœuds peuvent être associés à des catégories grâce à l'utilisation de labels. Une telle représentation des pistes cyclables a les avantages d'avoir une structure dynamique, un schéma flexible et d'effectuer des requêtes complexes avec une bonne performance.

Celle-ci nous permettra donc de connecter les pistes cyclables entre elles, à l'aide des distances qui les séparent les unes des autres d'une part et des points géographiques en commun d'autre part. Nous pourrons ainsi parcourir cette base de données par graphes pour réaliser des itinéraires possibles entre différents points géographiques. Finalement il sera possible de développer un algorithme permettant de choisir le meilleur itinéraire.

3. Détails du processus ETL

Dans cette section, nous présenterons l'extraction des données puis les étapes de transformations. Une vue à plus haut niveau est présentée à l'illustration 1.

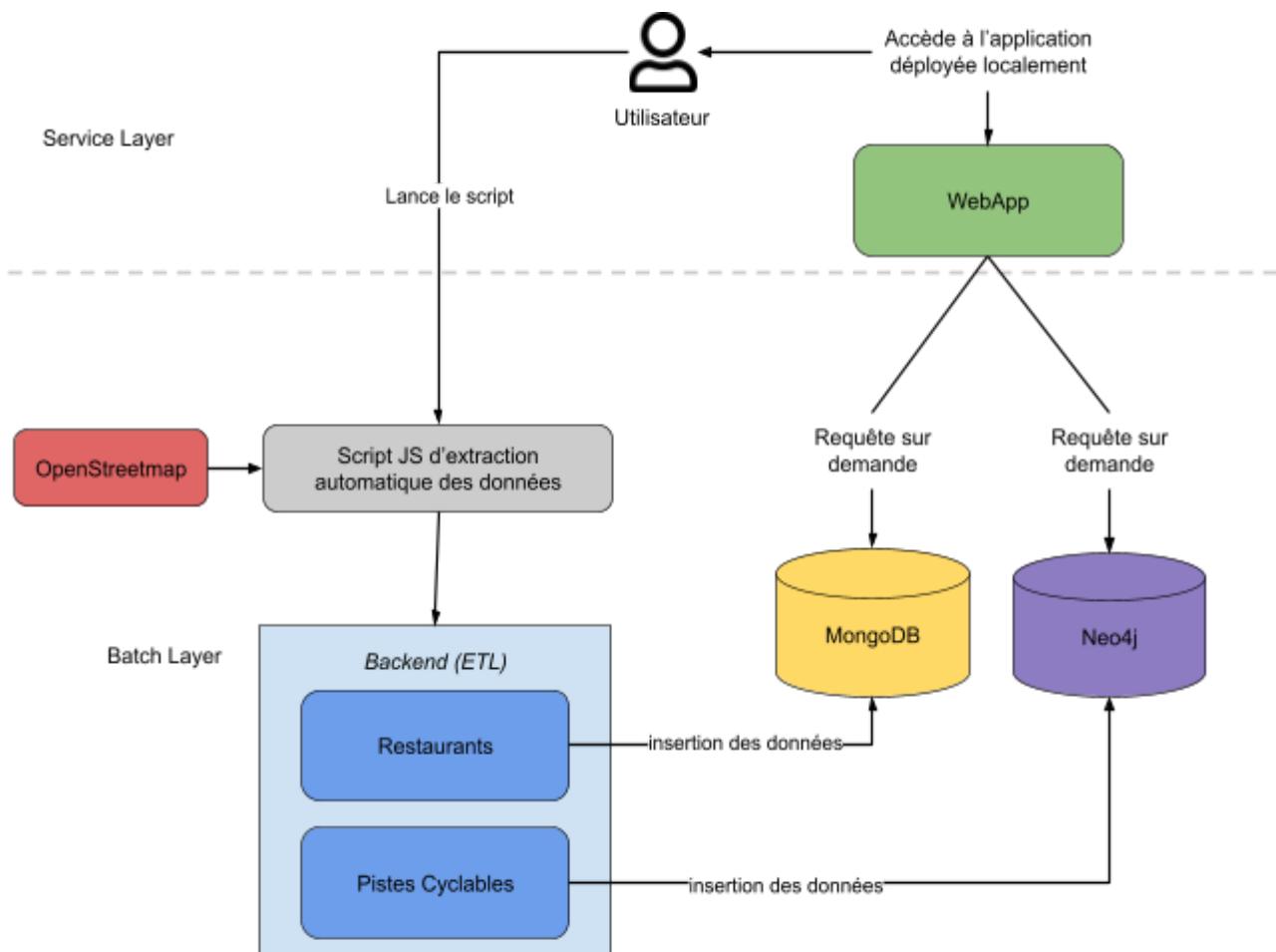


Illustration 1 : Architecture du flow des données

3.1. Obtention des données initiales et incrémentales

La première étape est d'obtenir les données de la source OpenStreetMap et de les intégrer dans les bases de données de l'application. L'illustration 2 présente ce processus.

Initialement, un utilisateur lance un script d'extraction des données, à la fois pour les restaurants et les pistes cyclables d'OpenStreetMap et les stocke dans un disque dur. Nous obtenons ainsi deux fichiers correspondants à la liste des restaurants et des pistes cyclables tel que présenté à la

section 1.2. Ensuite, pour chacun de ces fichiers, un second script va venir mettre en forme les données. Chaque donnée contient un identifiant OpenStreetMap, nous pouvons ainsi, lors de la mise à jour des données, vérifier si cet identifiant est présent ou non dans nos bases de données. Cette vérification d'identification permet une acquisition incrémentale des données.

3.2. Transformation des données

Pour la transformation des données des restaurants, nous avons fait en sorte de normaliser les types de restaurants. Chaque restaurant possède ainsi une liste de type de cuisine contenant au moins un élément. Cette liste correspond au champ **cuisine**. Un exemple de cette transformation est fourni en annexe à la section 4.2.1.

Pour la transformation des données de pistes cyclables nous allons simplement garder les tags correspondants à chaque route pour les relations des routes et on a créé un champ **idOSM** qui correspond à l'identifiant OpenStreetMap de la route. De plus, nous avons créé un champ **distance** pour chaque relation “Route” entre chaque point géographique qui correspond à la distance en mètre de chaque portion qui connecte deux points. Chaque point de la route correspondra donc à un point dans la base de donnée Neo4j avec uniquement les coordonnées géographique c'est-à-dire les champs **long** et **lat**.

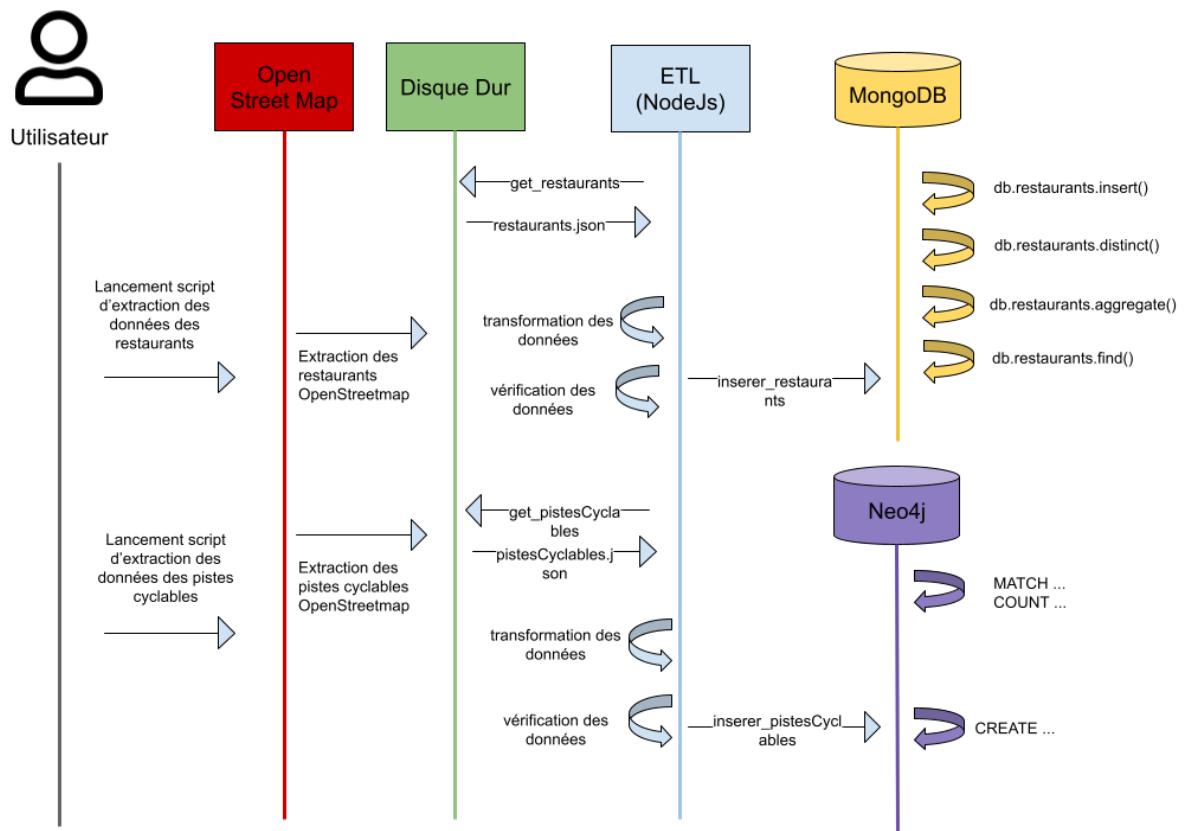
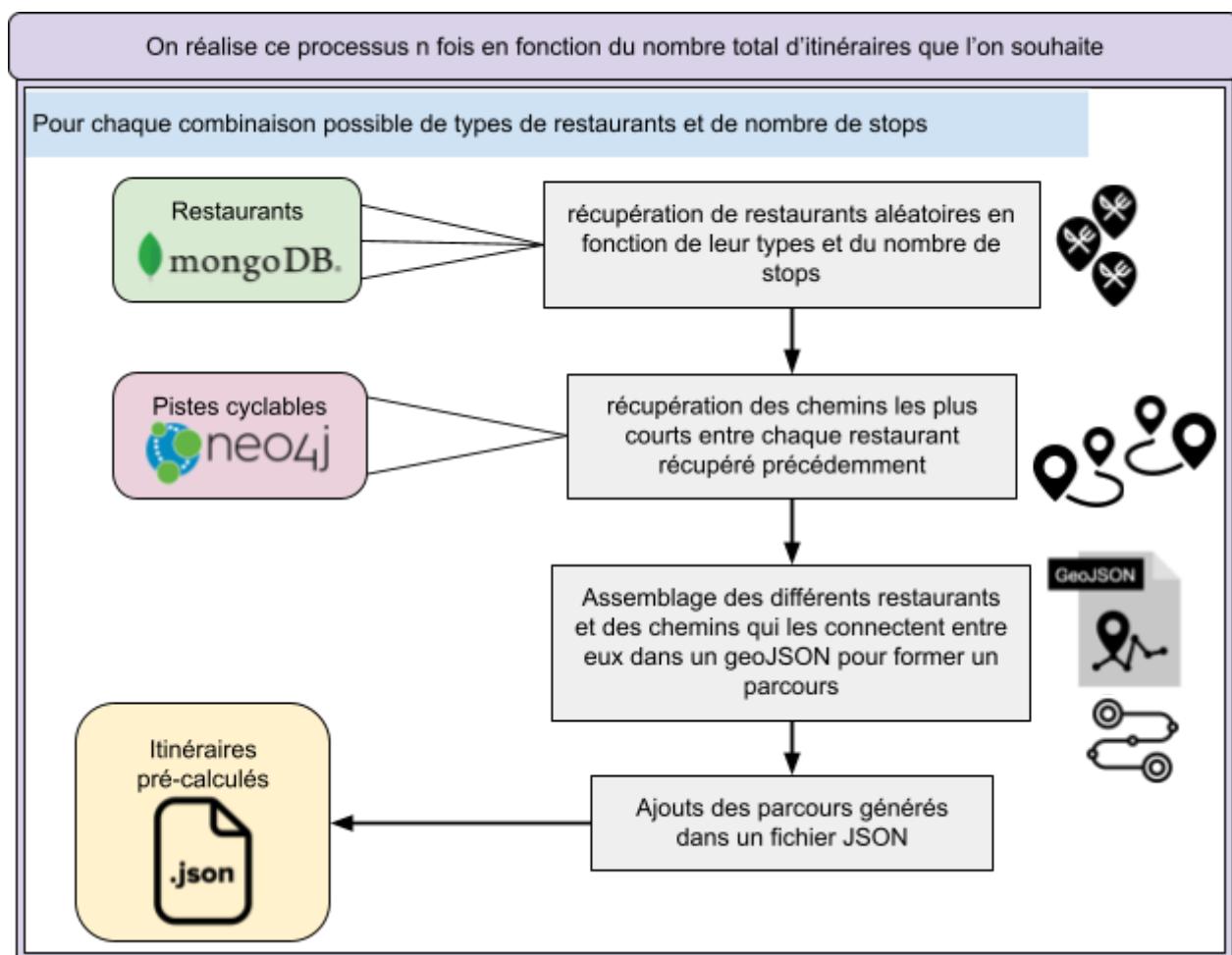


Illustration 2 : Processus d'extraction de l'application

4. Détails de la pipeline des données

4.1 Création d'un parcours

Pour la création des parcours, nous avons choisi une approche qui se base sur la génération d'un nombre important de parcours de manière aléatoire avec des configurations différentes. Nous allons ainsi venir générer n fois des parcours ayant des types de restaurants différents et des nombre d'arrêts différents. Voici le diagramme de séquence qui montre les différents flux de données lors de la création de nos parcours.



Nos parcours générés sont ainsi extrêmement précis entre les restaurants et quasiment optimaux. Pour la répartition des chemins les plus courts nous avons utilisé la fonction shortest path de Neo4j, qui permet de récupérer les transitions les plus courtes entre deux nœuds. Nous obtenons ainsi un itinéraire dans lequel le nombre de connexions entre des points géographiques pour relier deux restaurants est minimum. Cela nous donne ainsi des itinéraires qui vont privilégier les voies directes, plutôt avec les grands axes, ce qui correspond généralement à l'itinéraire le plus court en termes de temps également. De plus, nos données récupérées correspondent à toutes les routes que peuvent emprunter un vélo et les pistes cyclables. Le fait d'avoir plus de 70 000 points géographiques connectés les uns aux autres nous donne ainsi un très bon itinéraire entre deux restaurants.

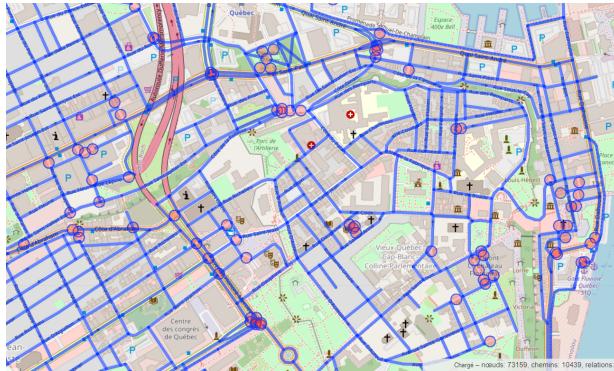


Image montrant les détails de nos données de routes utilisées dans notre application

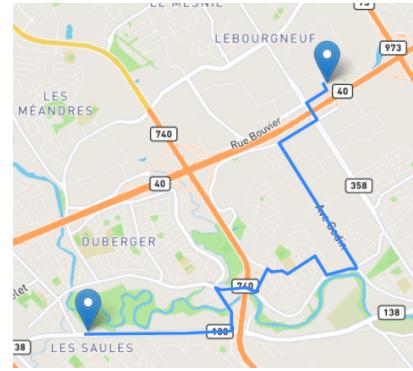
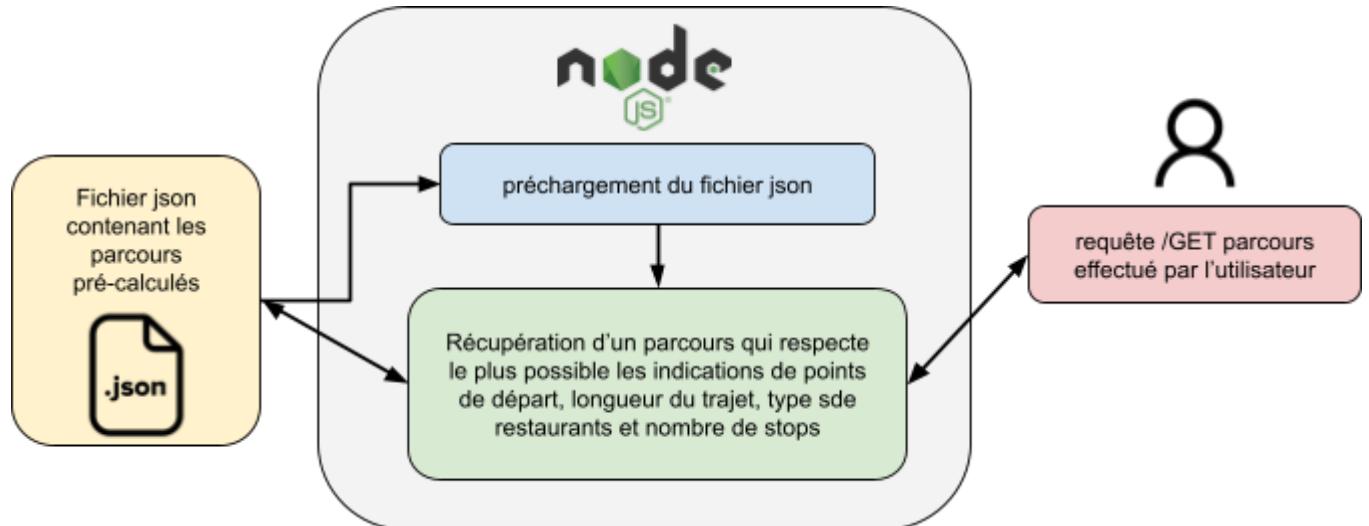


Image montrant un itinéraire entre deux restaurants généré par notre algorithme

4.2 Obtention d'un parcours

Pour l'obtention du parcours nous avons développé un système permettant de récupérer efficacement et très rapidement un parcours désiré en fonction des paramètres de la requête. Étant limité par la contrainte de temps de 1000ms pour l'obtention d'un parcours ainsi que la contrainte de temps de 30s pour le docker compose, nous avons fait le choix de stocker tous nos itinéraires pré-calculés dans un fichier json. Ce fichier json sera alors chargé par le serveur NodeJS lors de son lancement et dans lequel nous pourrons venir chercher rapidement et efficacement le parcours souhaité.



Comme montré dans ce diagramme, la procédure de récupération paraît assez simple. Cependant l'étape de récupération d'un parcours est lui assez complexe. Il faut arriver à sélectionner un parcours qui respecte le plus possible les conditions que l'utilisateur a définies lors de sa requête. Il faut également gérer les différents cas d'erreurs, afin de retourner les messages d'erreurs appropriés à l'utilisateur.

5. Explication du plan d'expansion

Pourquoi se fier à notre application ? Comment peut-elle gérer des données plus importantes ? Étant donné l'efficacité de notre algorithme, se limitant à la ville de Québec, nous souhaitons proposer notre application dans plusieurs villes du monde entier et permettre ainsi l'accès à notre application par beaucoup plus de personnes. Cela implique donc un volume plus important de données ainsi qu'une augmentation du nombre de requêtes par seconde effectuées par les utilisateurs. L'impact sera quantifiable en fonction des n nombre de villes supporté par notre application et également en fonction de la popularité de celle-ci.

5.1. Gestion d'un volume plus grand de données et d'utilisateurs

Pour permettre l'utilisation de notre application, nous devrons alors gérer des données plus importantes et un volume plus important d'utilisateurs. Pour cela, il est indispensable de distribuer les données.

5.1.1. Stratégie de réPLICATION

Tout d'abord, afin d'améliorer la fiabilité de notre application et s'assurer que les données soient accessibles rapidement et sécurisées, nous devons répliquer les données. Nous souhaitons pour cela utiliser une stratégie multi-leaders. Ainsi, plusieurs serveurs accepteront la charge relative aux écritures. Le fait d'avoir plusieurs chefs permettra d'améliorer les performances de notre application et de résister aux pannes. Pour le moment, notre application ne présente pas de cas où des données sont écrites par les utilisateurs mais on pourrait imaginer dans le futur de pouvoir stocker ces itinéraires en tant qu'itinéraires favoris ou encore d'actualiser les données en temps réel avec l'API OpenStreetMap. En cas de panne d'un chef, un autre chef sera élu parmi les suiveurs et les utilisateurs pourront ainsi toujours avoir accès à l'écriture de données. La réPLICATION des données nous permettra également de dupliquer nos données à différents endroits du globe pour ainsi mieux les sécuriser au cas où un serveur viendrait à avoir une panne matérielle. De plus, la donnée étant répliquée à plusieurs endroits, la charge serveur sera répartie sur l'ensemble des multiples serveurs et ainsi le nombre de requêtes par seconde pourra être géré.

5.1.2. Stratégie de partitionnement

Ensuite, pour répondre au besoin d'extensibilité et permettre d'effectuer les requêtes d'écriture et de lecture sur un grand volume de données, nous souhaitons en parallèle de cette réPLICATION, distribuer les données en les fragmentant sur les différents serveurs. Nous souhaitons partitionner les données de manière horizontale. La quantité de données sera partagée équitablement sur chaque nœud, composé d'un chef et de suiveurs, pour permettre un équilibre. Il est évident que notre application sur le long terme aura certaines données plus demandées que d'autres. Nous pouvons imaginer par exemple qu'une ville où beaucoup de personnes utilisent leur vélo comme Amsterdam demandera plus de ressources qu'une ville qui utilise moins le vélo comme Québec. Nous pourrions ainsi définir un nombre de fragments dynamiquement en fonction de la charge de requêtes effectué sur le nœud et répartir chaque nœud avec des fragments de villes dont la

demande est différente afin de répartir le volume de requêtes. Nous créerons ainsi des fragments pour les parcours de chaque ville ou ensemble de villes.

5.1.3. Sécurité des bases de données

Pour empêcher les attaques externes sur notre base de données et s'authentifier, nous procéderons de la manière suivante. Etant donné que notre application repose sur MongoDB, une base de données NoSQL dont la configuration par défaut n'est pas sécurisée, elle permet une utilisation en ligne sans authentification, au risque de perdre des données. Pour pallier ce problème, nous activerons l'authentification. Cela permettra de contrôler les priviléges des rôles. Les administrateurs de la base de données pourront modifier les données de la base de données, par exemple ajouter des données relatives à d'autres villes. Un utilisateur ne pourra que lire les collections de MongoDB et les données de Neo4j. Nous attribuerons ensuite un rôle avec certains droits d'écriture si l'utilisateur est identifié et connecté à notre application. Nous pourrons ainsi dans le futur ajouter des parcours favoris à son profil par exemple. A chaque authentication d'un utilisateur, un token à durée de vie limitée sera transmis à l'utilisateur et il aura besoin de ce token pour effectuer toutes les autres requêtes. C'est-à-dire que tout utilisateur de notre application devra utiliser un token en *header* pour effectuer chacune des requêtes.

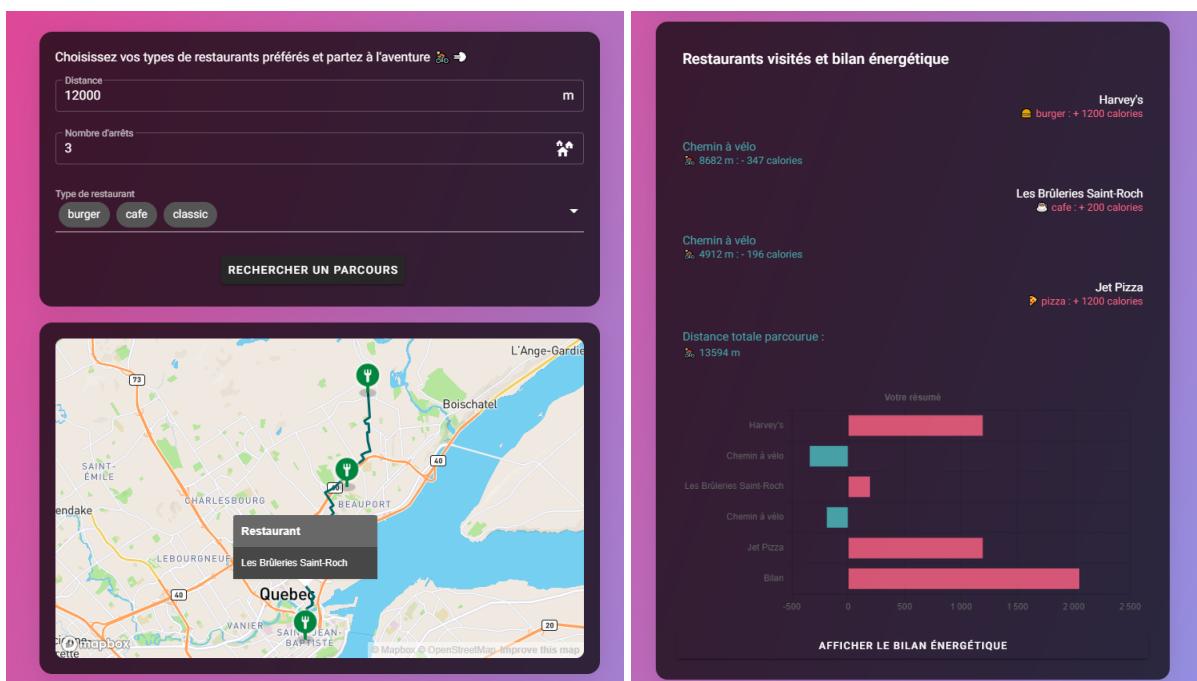
Pour la sécurité de la base de données nous pourrons créer des backup régulier à différents endroits de la planète avec des données plus ou moins à jour (par exemple -1 jour et -1 mois) pour ainsi sécuriser nos données si celles-ci venaient à être corrompues. Nous pourrions alors imaginer, en cas d'attaque du système ou de problème technique de celui-ci, pouvoir récupérer des données relativement à jour sans trop d'impact sur celui-ci.

5.2. Fonctionnalités additionnelles pouvant être implémentées grâce à la structure des données

A l'heure actuelle, les requêtes que nous effectuons sont particulièrement rapides. Ainsi, notre API peut facilement s'intégrer dans une application que tout le monde souhaiterait utiliser au quotidien. La structure des données actuelle peut être ainsi implémenter pour potentiellement toutes les villes du monde entier. La seule limite de notre système actuel est la validité des données que l'on peut trouver sur OpenStreetMap. Les données d'OpenStreetMap étant open source, il est nécessaire que celles-ci soient mises à jour par la communauté. De plus, notre application se limite à 5 types de restaurants pour ne pas surcharger la taille de l'application mais celle-ci possède l'entièvre capacité de pouvoir être déployée avec bien plus de restaurants et n'est absolument pas limitée par ces 5, bien au contraire. Nous pourrons également dans le futur pouvoir greffer de nouvelles données à notre application, comme par exemple un système d'évaluation, de notes de restaurants ou bien encore des données géographiques plus détaillées avec des données d'altitudes pour voir le dénivelé effectué en vélo, qui est une information intéressante et pertinente. Nous pourrions également greffer à nos données un système de recommandation d'itinéraires basé sur les itinéraires que l'on a effectués précédemment.

6. Fonctionnalités avancées

Nous avons souhaité réaliser une interface graphique pour notre application qui permet ainsi à l'utilisateur d'effectuer des requêtes en temps réel avec l'appel à notre API. Cette interface permet à l'utilisateur de choisir un nombre d'arrêts maximum de restaurants qu'il souhaite visiter, une distance à parcourir à vélo et les types de restaurants qu'il souhaite visiter. Si notre utilisateur aime se dépenser et se faire plaisir, tout en contrôlant son alimentation, il peut aussi, avec l'interface web de notre application, obtenir des informations sur sa dépense énergétique entre chaque restaurant, en fonction de la distance qu'il parcourt à vélo. Il peut aussi savoir combien ce qu'un repas type d'un fast-food, d'un café, d'une pizzeria ou encore d'un restaurant classique, apporte comme apport nutritif et ainsi évaluer le bilan de sa journée. Pour nous, il était inconcevable que notre application ne comporte pas d'interface graphique, celle-ci permet de véritablement visualiser l'itinéraire et de générer celui qui nous plaît.



Nous notons également la présence de nouvelles données caloriques pour à la fois la distance parcourue en vélo mais également pour chaque type de restaurant.

Une autre fonctionnalité que nous pouvons définir comme étant avancée est l'utilisation de données de routes et chemins accessibles en vélo. Notre application ne possède pas uniquement certains trajets prédéfinis sur des pistes cyclables d'une certaine longueur. Nous avons une réelle connexion entre tous les points géographiques et routes/chemins accessibles au vélo. Comme GoogleMap, notre base de données vient reconstruire un itinéraire qui est lui extrêmement précis. Cette extrême précision sur l'itinéraire engendre un nombre important de points (78 000 points connectés entre eux dans la base de données neo4j). Ce nombre important de points connectés les uns aux autres n'a pas été évident à réaliser, mais permet ainsi de pouvoir connecter n'importe quelle position sur la ville de Québec avec n'importe quelle autre position.

7. Annexes

7.1. Exemples de données

7.1.1. Données de restaurants

```
[out:json];
{{geocodeArea:"Quebec City"}}->.searchArea;
(
  node["amenity"~"restaurant|fast_food|cafe"](.searchArea);
  way["amenity"~"restaurant|fast_food|cafe"](.searchArea);
  relation["amenity"~"restaurant|fast_food|cafe"](.searchArea);
);
out center;
out body;
>;
out skel qt;
```

Figure 1 : Requête overpass, qui permet d'extraire les données des restaurants, fast-foods ou cafés situés dans la ville de Québec.

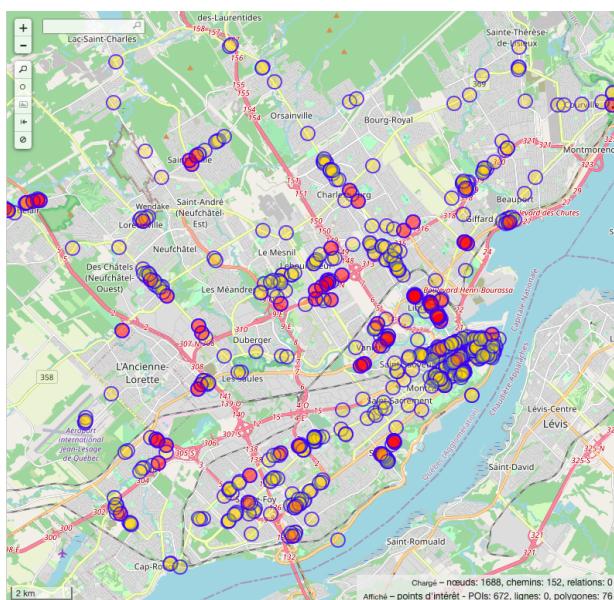


Figure 2 : Résultats cartographiés de la requête effectuée en figure 1.

```
{
  "type": "node",
  "id": 291667428,
  "lat": 46.8532654,
  "lon": -71.1969194,
  "tags": {
    "addr:housenumber": "411",
    "addr:street": "Boulevard Sainte-Anne",
    "amenity": "fast_food",
    "brand": "Subway",
    "brand:wikidata": "Q244457",
    "brand:wikipedia": "en:Subway (restaurant)",
    "cuisine": "sandwich",
    "name": "Subway",
    "takeaway": "yes"
  }
}
```

Figure 3 : Exemple d'un point de données de restaurant, résultat de la requête effectuée en figure 3 au format JSON.

7.1.2. Données de routes

```
area["name:en"="Quebec City"]->.searchArea;(
way[highway~^(trunk/primary/secondary/tertiary/unclassified/residential/Living_street/road/pedestrian/path/bridleway/cycleway$)][area!=yes](area.searchArea);
way[highway=footway][bicycle=yes](area.searchArea););out body;>;out skel qt;
```

Figure 4 : Exemple d'une requête overpass, qui permet d'extraire les données des pistes cyclables passant dans la ville de Québec.

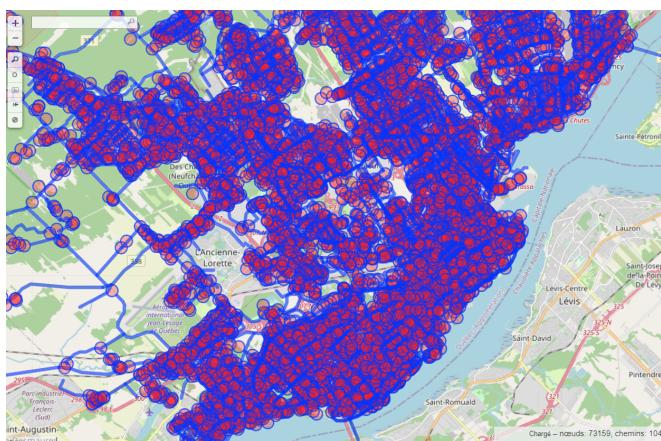


Figure 5 : Résultats de la requête effectuée en figure 4 cartographiés.

```
{"type": "Feature",
  "properties": {
    "@id": "relation/215725",
    "name": "Jonction Aquarium - Promenade Samuel-De-Champlain",
    "network": "Lcn",
    "route": "bicycle",
    "type": "route"
  },
  "geometry": {
    "type": "MultiLineString",
    "coordinates": [
      [[ -71.29218, 46.7520642],
       [-71.2920145, 46.7520578],
       ....],
      ...
    ],
    "id": "relation/215725"
  }
}
```

Figure 6 : Exemple d'un point de données de route, résultat de la requête effectuée en figure 4, au format GeoJSON.

7.2. Exemples de transformation des données

7.2.1. Transformation des données de restaurants

Données initiales

```
{
  "type": "Feature",
  "id": "way/190858158",
  "properties": {
    "type": "way",
    "id": 190858158,
    "tags": {
      "addr:housenumber": "5146",
      "addr:postcode": "G2E 2G9",
      "addr:street": "Boulevard Wilfrid-Hamel",
      "amenity": "restaurant",
      "building": "yes",
      "building:levels": "2",
      "cuisine": "italian_pizza;diner;pasta;breakfast",
      "internet_access": "wlan",
      "name": "Normandin",
      "opening_hours": "Mo-We 05:30-01:00; Th-Fr
05:30-03:00; Sa 06:00-03:00; Su 06:00-01:00",
      "operator": "Normandin",
      "phone": "+1-418-877-1192",
      "website": "https://www.restaurantnormandin.com/"
    },
    "relations": [],
    "meta": {}
  },
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        [
          [
            [
              [
                [
                  [
                    [
                      [
                        [
                          [
                            [
                              [
                                [
                                  [
                                    [
                                      [
                                        [
                                          [
                                            [
                                              [
                                                [
                                                  [
                                                    [
                                                      [
                                                        [
                                                          [
                                                            [
                                                              [
                                                                [
                                                                  [
                                                                    [
                                                                      [
                                                                        [
                                                                          [
                                                                            [
                                                                              [
                                                                                [
                                                                                  [
                                                                                    [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
................................................................
```

Données transformées

```
{
  "type": "Feature",
  "id": "way/190858158",
  "properties": {
    "type": "way",
    "id": 190858158,
    "tags": {
      "addr:housenumber": "5146",
      "addr:postcode": "G2E 2G9",
      "addr:street": "Boulevard Wilfrid-Hamel",
      "amenity": "restaurant",
      "building": "yes",
      "building:levels": "2",
      "cuisine": [
        "italian",
        "pizza",
        "diner",
        "pasta",
        "breakfast"
      ],
      "internet_access": "wlan",
      "name": "Normandin",
      "opening_hours": "Mo-We 05:30-01:00; Th-Fr
05:30-03:00; Sa 06:00-03:00; Su 06:00-01:00",
      "operator": "Normandin",
      "phone": "+1-418-877-1192",
      "website": "https://www.restaurantnormandin.com/"
    },
    "relations": [],
    "meta": {}
  },
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        [
          [
            [
              [
                [
                  [
                    [
                      [
                        [
                          [
                            [
                              [
                                [
                                  [
                                    [
                                      [
................................................................
```

Figure 7 : Exemple d'un point de données de restaurant

Figure 8 : Transformation de ce point de données

7.2.2. Transformation des données de pistes cyclables

Données initiales

```
{
  "type": "Feature",
  "id": "way/50249013",
  "properties": {
    "type": "way",
    "id": 50249013,
    "tags": {
      "highway": "primary",
      "name": "Boulevard Sainte-Anne",
      "oneway": "yes",
      "ref": "138"
    },
    "relations": [],
    "meta": {}
  },
  "geometry": {
    "type": "LineString",
    "coordinates": [
      [
        [
          -71.1500697,
          46.8816954
        ],
        ...
      ]
    ]
  }
}
```

Figure 9 : Exemple d'un point de données de pistes cyclables

Donnée d'un point géographique faisant partie d'une route

```
{
  "identity": 1,
  "labels": [
    "Point"
  ],
  "properties": {
    "long": -71.1524423,
    "lat": 46.8806688
  }
}
```

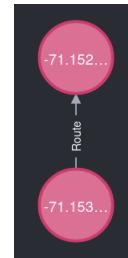


Figure 10 : Exemple d'un point géographique

Figure 11 : Exemple de deux noeuds de la base Neo4j connectés par une relation "Route"

Donnée transformée d'une relation "Route" entre deux points

```
{
  "identity": 38779,
  "start": 36264,
  "end": 0,
  "type": "Route",
  "properties": {
    "ref": "138",
    "distance": 79.79500962362903,
    "idOSM": 50249013,
    "name": "Boulevard Sainte-Anne",
    "highway": "primary",
    "oneway": "yes"
  }
}
```

Bibliographie

- [1] *Overpass API/Overpass API by Example* - OpenStreetMap Wiki. (s. d.). Documentation Overpass. Consulté le 25 septembre 2021, à l'adresse https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_API_by_Example
- [2] *Use Docker Compose.* (2021, 24 septembre). Docker Documentation. https://docs.docker.com/get-started/08_using_compose/