UNIVERSITÉ LAVAL

Faculté des sciences et de génie

Département d'informatique et de génie logiciel

GLO-7035

Base de donnée avancé

Remise 4

Session : Automne 2019

Presenté à :

M. Jean-Thomas Baillargeon

Par l'équipe 2 :

Jonathan Ouellet : 111 125 752

Maxime Lapointe : 909 158 759

Tables des matières

1. Présentation des données

Cette section présente les données brutes acquises dans le cadre du projet, leurs utilités, ainsi que quelques faits saillants.

1.1 Source, taille & analyse descriptive

Les données utilisées sont pour la ville de Québec. Les éléments de base des pistes cyclables proviennent des données ouvertes de la ville de québec. Ces dernières sont téléchargeables dans plusieurs formats, dont le format GeoJson. Contrairement aux données d'autres villes, comme celle de Montréal, ces pistes comportent aussi les voies désignées (en bordure de route), en plus des pistes spécialement aménagées. Elles offrent des distances approximatives, des directions, ainsi que le nom de la route. Le tout pèse tout au plus quelques mégaoctets et offre un total de 3245 objets de type *linestring*, de longueurs variables. La distance totale estimées par des formules spécialisées en géodésiques est de 485.69km de route cyclable (les données, comme l'application, utilise le standard *World Geodesic Survey 1984 (WGS84)*, un standard mondial en coordonnées GPS). Les routes ne se chevauchent pas, mais s'intersectent à quelques reprises, en milieu de segments.

Les données de restaurants proviennent de *Openstreetmap*. L'api permet d'aller chercher directement tous les restaurants d'une ville en particulier et leurs coordonnées. Cet ensemble contient un total de 453 restaurants dans la grande région de la Capitale nationale. Ces données offrent aussi des caractéristiques intéressantes additionnelles, comme la *brand* (les chaînes de restos), le nom des restaurants, la "cuisine" présente dans ces restaurants (ex: pizza, pâtes), le type de bâtiment. Malheureusement, les champs sont entrés à la main par les utilisateurs et optionnels, ce qui en fait un champ non-standardisé et non-indicatif du type du restaurant. Pour ce faire, l'équipe a dû créer un champ "type" standard avec des catégories de restaurant, comme "pub", "buffet", "fastfood", etc. Cet ajout est discuté en détail dans la section de modélisation des données.

Pour ce qui est de données additionnelles, l'équipe est allé trouver des critiques (pour offrir des fonctionnalitées avancées). Ces dernières proviennent d'un concours *Yelp* du traitement du

langage naturel comportant des critiques de restaurants. Les données de ce type sont normalement générateur d'une valeur importante pour les compagnies et sont rarement ouvertes et téléchargeable, point qui sera discuté davantage dans le plan d'expansion.

Un échantillon a été conservé (la taille originale fait plusieurs gigaoctets) et distribué aléatoirement entre les restaurants de Québec, puisque les données et reviews proviennent en quasi-totalité des États-Unis. Ces reviews offrent des critiques sous forme de texte, ainsi que le score attribué à la critique par le reviewer en étoile (1 à 5). Ils proviennent de plusieurs langues, 98% étant en anglais, mais certaines étant en espagnol, français, et autres langues.

2. Technologies utilisées pour le projet

Cette section commence par décrire les langages et bases de données (BD) utilisées pour le projet, ainsi que la logique derrière ces choix, suivit des méthodes pour transformer les données brutes en données utiles et les insérer.

2.1 Langage de programmation et bases de données

Le langage de programmation utilisé dans l'application est Python. Une attention particulière à été utilisées afin que tous les éléments utilisent le même langage, afin de limiter les risques de frictions entre divers scripts et différentes opérations reliant diverses parties du fonctionnement interne. Puisque nos fonctionnalitées avancées requièrent du traitement de la langue naturel et que le plan d'expansion nécessite davantage de data engineering, de data processing et l'ajout de modèle avancé en data science (dont l'intelligence artificielle), des langages comme javascript et C++ ont été rapidement écartés, puisque la plupart des librairies utilisés dans ces domaines sont en python.

De ce fait, les pilotes utilisés sont pymongo, ainsi que py2neo, pour communiquer avec les bases de données MongoDB et Neo4j, respectivement. Le framework web utilisé pour l'application est Flask, qui est lui aussi codé en python. Il sert principalement à gérer les communications http entres les utilisateurs et les bases de données.

Pour l'instant, l'application n'utilise pas de base de données ElasticSearch pour les reviews et de driver python pour les requêtes dans cette base de donnée. Ce choix a été fait puisque les fonctionnalitées de l'application n'offrent pas encore de recherche dans le texte des reviews et que les reviews sont tous disponibles et calculés à l'avance. Ce choix sera à revoir très rapidement advenant une expansion de l'application (plus de détails à la section 5).

2.2 Modélisation des données collectées en BD

2.2.1 Restaurants et critiques

Pour finaliser le fichier de données des restaurants (incluant les données de critiques), des données on été ajoutées au GeoJson des restaurants de Québec. Ces dernières incluent: une liste des reviews, une liste correspondant à la probabilité que ces critiques soient positive, un type de restaurant et un champ moyenne des critiques par restaurant.

- Type

Comme il n'y avait aucune attente précise sur les types de restaurants et aucun standard défini, un ensemble de cinq types temporaires (la taille comme les noms des types sont facilement interchangeables) a été créé: *fastfood*, *5 étoiles*, *classic*, *buffet* et *pub*. Ces derniers ont été attribué aléatoirement entre les restaurants, mais de manière à éviter que deux restaurant ayant le même nom offrent des types différents.

- Liste des reviews

La liste des reviews est celle mentionnée dans les données collectées. On parle ici des reviews sous forme de texte, mais sans les notes générées par les utilisateurs.

- Probabilité de critique positive

Ces probabilités requièrent un modèle afin d'extraire, à partir des reviews en texte, et des scores utilisateurs (en étoiles), une probabilité que le restaurant soit "bon". Les détails sur ce processus est dans la section des fonctionnalitées avancées. Cette valeur varie entre 0 et 1, 0 indiquant le pire des restaurants et 1, le restaurant parfait.

- Moyenne des critiques par restaurant

Puisque les critiques sont actuellement statiques, un champ contenant le score moyen des critiques par restaurant a aussi été créé, afin de simplifier les requêtes Neo4j. Advenant le cas que les critiques peuvent être modifiées, ajoutées, retirées ou autres, ce champs devra être retiré et entièrement géré par les requêtes utilisant les champs de score de critique.

2.2.2 Obtenir les statistiques des routes

Les routes, dans l'application actuelle, n'offrent à l'utilisateur que la distance parcourue. Les autres données dans le fichier téléchargé ne sont pas encore valorisées, principalement dû à la technique actuelle de gestion du graphe des routes. La distance est calculée, pour chaque segment (défini par une paire de point avec longitude et latitude), via des formules de géodésiques avancées. Ces formules tiennent compte du fait que la terre n'est pas une sphère mais un ellipsoïde désaxé (standard WGS84), ce qui fournie une précision dans l'ordre du mètre. Ce choix a été fait, compliquant les calculs comparativement à l'assomption sphérique de la terre, puisque l'erreur d'estimation grandi avec la latitude. Avec les mesures elliptiques, même avec des latitudes élevées, comme c'est le cas au Canada, il est possible d'arrondir au mètres prêt sans trop de problème.

2.2.3 Gérer un graphe de routes cyclables

Dans l'éventualité que certains segments se croisent, l'équipe a prévu un algorithme utilisant les standard WGS84, qui compare chacune des lignes (lineString ou multiLineString) avec les autres et qui vérifie s'il y a des intersections.

Dans les faits, l'algorithme divise les lignes en segments (paires de points reliées), les empiles (dans un *stack*). Il *pop* un segment et le compare avec tous les autres segments pour vérifier s'il y a un intersection. Ces opérations sont fait dans la mongoDB, et injecte les segment appropriés dans la Neo4j sous forme de graphe.

Les segments ajoutées dans la Neo4j sont deux nodes de type "Intersection", liés par une relation de type "Connects_to" (la direction n'est pas encore liées aux sens uniques des

données). Les nodes contiennent les propriétées de points (coordonnées GPS) et les relations la propriété distance calculées par géodésique.

Voici le fonctionnement de l'algorithme, qui forme le graphe à partir des segment contenu dans la mongoDB:

Pour tous les segments:

- Sans intersection: Le segment est ajouté à Neo4j
- Si Intersection: regarder toutes les intersections, en ordre:
    - L'algorithme trouve le point exact d'intersection en longitude et latitude, par un calcul utilisant des *great circles* (loxodromes) et des *initial bearings*.
        - Précision au mètre prêt. La plupart des applications géographique utilisent des *rhumb lines* (orthodromes) pour le calculs des distances géospatiales, ce qui auraient réduit la précision au delà du mètre pour des distance supérieures à 1km. L'équipe a jugé que, compte tenu du mode de déplacement, des erreurs d'une dizaine de mettre sur des trajets d'un seul kilomètre était à la limite du "passable".[1]
    - Si les intersection sont toutes aux bout d'un des segments (les points), ceci ne fait qu'indiquer que les segment sont bout à bout, et qu'il n'y a rien de précis à faire (lignes continues).
        - On ajoute le segment dans la Neo4j
    - Lorsque une ou plusieurs intersections ne sont pas aux extrémitées
        - Algorithme ajoute un point intersection avec les bonnes coordonnées
        - Forme des nouveaux segments en liant les points des segments et le nouveau point
            - Afin de limiter l'explosion des connections.
            - Pour le moment, cette fonction fait quelques assomptions qui ne minimisent pas encore le nombre de connection, ce qui ferait une fonctionnalité intéressante pour donner des parcours de qualitée supérieures.

[1] Un jupyter notebook est fourni avec la remise pour montrer les fonctions de calculs par loxodromes et bearings. L'équipe considère ces mathématiques comme une fonction avancée qui apporterait une valeur additionnelle importante si joint à d'autres fonctionnalitées.

- Ajoute les nouveaux segments dans la neo4j

2.2.4 Insérer les restaurants

L'insertion des restaurants dans la Neo4j, se fait en lisant le GeoJson bonifié (le même qui est utilisé pour . Le script crée une node de type "Restaurant" contant un id, un nom, un type, une liste de critique, une liste de score de critique, la moyenne de ceux-ci, ainsi que des coordonnées en latitude et longitude.

2.2.5 Lier les restaurants aux routes

L'équipe a fait le choix d'utiliser une requête très simple, mais qui s'avère tout de même efficace. Le but est de lier les nodes restaurant de la Neo4j aux nodes Intersection proches (géographiquement) du restaurant.

La technique choisie est d'itérer sur tous les restaurants dans notre base de donnée MongoDB (qui ont leur équivalent unique dans la Neo4j). et de comparer à tous les points formant les segments existant dans la MongoDB (qui ont aussi leur équivalent unique dans la Neo4j).

Cette comparaison est une requête $nearSphere, de $maxDistance 70 mètres (valeur arbitraire choisies pour avoir au moins 2 connections de chaque restaurant à des points d'intersection de routes).

Pour chaque extrémité de segment a proximité du restaurant, récupère les nodes correspondantes "Restaurant" et "Intersection" crée une relation "Connects_to" qui les lient.

3. Algorithme permettant de produire les parcours

Cette section décrit comment les différentes bases de données interagissent et comment les parcours sont déterminés

3.1 Communication des valeurs entre les bases de données

Dans l'application, le seul moment où les bases de données communiquent entre elles est lors du remplissage de la base de donnée Neo4j. Tout ce processus a été décrit, dans l'ordre, à la section 2. L'importation profite de la puissance des requêtes géospatiale de MongoDB pour remplir la Neo4j. Les appels pour la génération des données utilisent donc, l'une ou l'autre des bases de données selon l'aisance à faire les transformations.

Une fois les données mises en place, il n'y a plus de raison que les BD communiquent entre elles. Les requêtes de statistique (comme /heartbeat) de route utilise la mongoDB, alors que les requêtes basées sur les restaurant et les trajets utilisent la Neo4j, car les informations utiles de restaurants se retrouvent directement dans les nodes.

3.2 Calculs permettant de trouver le parcours épicurien le plus intéressant pour l'usager

3.2.1 Les routes!

Notre application offre 3 types de parcours épicurien. Dans l'ordre, il y a le parcours "standard", le parcours "sans duplicata" et le parcours "du critique". Chacun de ces parcours "construit" par dessus son prédécesseur, pour affiner la recherche et avoir un parcours plus intéressant pour l'utilisateur. Le parcours du critique utilise les fonctionnalitées avancées.

Du point de vu utilisateur, la différence est dans les types de parcours reçus, mais aussi avec les critères (demandes) que l'utilisateur envoi à l'application.

Le parcours standard demande un point de départ (qui peut être trouvé grâce à la fonction *point de départ*) et des types de restaurants (trouvé avec la fonction *type*). Standard demande aussi une distance approximative pour le trajet demandé et le nombre d'arrêt (restaurants) idéal.

Le parcours sans duplicata ne fait que s'assurer que les retours n'offrent jamais le même restaurant, ce qui peut arriver dans le parcours standard, si les critères sont trop serrés. Il ne demande pas plus d'entrées de l'utilisateur.

"Manger au McDo deux ou trois fois dans une journée, c'est dangereux pour la santé!" - Louise Sirois, 2007.

Le parcours du critique, dans son cas, demande 3 paramètres de plus en entrée. Le minimum, le maximum et la moyenne. La moyenne s'assure que les indices moyens de qualités (entre 0 et 1) pour chaque restaurant dans le trajet soient, en moyenne supérieur à cette valeur. Le minimum permet de retirer les restaurant qui ont une cote moyenne de qualité plus bas que cette valeur et le maximum, permet de retirer les restaurants avec une cote moyenne de qualité supérieure à cette valeur.

Pourquoi mettre la cote maximale? Pour essayer des restaurants moins "mainstream" ou encore des restaurants moins "populaires" dans la population en générale. Les reviews inclus malheureusement des biais… par exemple le service peut être médiocre, mais la nourriture excellente!!!! Ce restaurant pourrait malheureusement avoir 0.4 comme indice! Un must pour les *foodies* qui veulent expérimenter par eux-même et former leur propre critique!

3.2.2 Les calculs

Le parcours standard commence par s'assurer de commencer dans les environ du point de départ sélectionné et de terminer à un restaurant inclus dans les types demandé! Rien de mieux que de terminer avec un restaurant ayant un style demandé et d'éviter de finir dans un buffet chinois après avoir mangé dans un resto 5 étoiles!

Il retirer ensuite les chemin qui ne sont pas dans les distances demandées. On veut bien faire du vélo, donc distance à vélo + ou - 10%!

Il vérifie qu'il y ait au moins un resto de chaque type dans la route complète et que le nombre de resto ne dépasse pas le nombre idéal.

Le parcours sans duplicata s'assure aussi de ne pas repasser par des restaurants qui porte le même nom!

Le parcours du critique, quant à lui, utilise des inégalitées sur les indices de critiques pour réduire davantage les restaurants admissibles dans le parcours (et aussi, sans duplicata)

Dans un but d'avoir des requêtes qui s'exécutent dans un délais raisonnable, et considérant que personne de l'équipe n'a de machine surpuissante, il a été décidé de limiter le nombre de saut à 20 connection à partir du départ.

Les parcours retournés sont ordonnées par nombre de stop (pour avoir le plus près du nombre demandé), puis par distance, puis par cote moyenne des resto (si c'est le parcours du critique seulement)

3.2.3 Performances limitées et appels rapides

Les requêtes vont chercher une intersection, correspondant à celle fournie par l'usager, à 20 sauts ou moins d'un restaurant dont le type doit faire partie de la liste fourni par l'usager. Le nombre de saut est un facteur qui, dû au fort nombre de connection entre les routes, d'être limité arbitrairement.

De plus, pour éviter les duplicatas, des opérations de *unwind* et *collect* doivent être effectuées, et seulement les 30 000 chemins potentiel (avant ordonnancement) sous soumis à la "dé-duplication", puisque cette étape prend plus de 10 fois le temps des étapes précédentes.

4. Documentation de l’api

Cette section décrit les endpoints, leurs utilités, leurs requis et leurs sorties.

Les endpoints de l’application sont les ceux qui étaient demandés, ainsi que quelques additions pour donner plus d’information ou de possibilités à l’utilisateur. En voici la description:

Le premier est “/heartbeat”, qui donne la quantité de restaurants, ainsi que la longueur totale en km du réseau de pistes cyclables.

Le endpoint “/readme” permet de télécharger un fichier de type markdown contenant la liste de endpoints avec des exemples de valeur de retour et de payload si nécessaire.

Il est possible d'accéder à “/type” qui retourne une liste de tous les types de restaurants contenu dans les données.

Le endpoint “/starting-point” nécessite un body de type json dans la requête. Il faut retrouver une liste de type pour les restaurant intéressants, ainsi qu’une distance désirée pour le trajet. Le endpoint retourne une intersection avec ses coordonnés.

Après avoir obtenu un point de départ, l’utilisateur peut accéder à “/parcours” en lui fournissant dans un payload un point de départ désiré, un nombre d’arrêt souhaité, la liste de type de restaurant ainsi que la longueur du trajet, ceci retournera une liste de segment contenant chacun un objet de type lineString, les informations du restaurant et le numéro du segment.

Il existe aussi un “/parcoursWithoutDuplicate” qui prend les mêmes paramètre d’entré et de sortie mais qui assure de ne pas avoir deux fois un même restaurant sur le trajet en contrepartie d’un temps de requête plus long.

Une troisième option, le endpoint “/parcours_du_critique”, s’offre aux utilisateurs. Il utilise des fonctionnalités avancées du traitement des critiques. En plus de prendre les paramètres

précédents, cette option demande un score minimal et maximal, ainsi qu'une moyenne minimale.

Une autre fonctionnalité qu'il est possible d'utiliser est celle de visualiser tous les critiques propres à un restaurant en utilisant "/review" et en envoyant le nom et coordonnées de restaurant en question. Pour faciliter cette requête, il est possible d'accéder à la liste de tous les restaurant avec un même nom via "/resto_list_by_name" et en fournissant le nom du restaurant d'intérêt.

5. Explication du plan d'expansion

Cette section détaille les données supplémentaires (provenant d'utilisateurs et provenant d'autres sources) pouvant être collectées et ajoutées dans l'application. Elle discute ensuite des fonctionnalitées potentielles de ces données et de leur intégration.

5.1 Gestion d'utilisateurs et données utilisateurs

5.1.1 Ajout d'utilisateurs

Une amélioration qui va de soi serait de permettre aux utilisateurs de s'authentifier pour être capable de lier certaines informations supplémentaire à leur profil, mais aussi de permettre aux utilisateurs d'entrer des données qui peuvent, lorsqu'en grande quantité, offrir de la valeur additionnelle pour les investisseurs, les fonctionnalitées, mais aussi les autres utilisateurs.

L'interface de l'application serait essentiellement un interface web (html, css, etc.) avec des méthodes pour lire les retours et envoyer les payloads fournis par les utilisateurs (frontend qui ne fait que collecter des demandes et afficher l'information).

L'application devrait être augmentées pour soutenir les informations de login des utilisateurs, qui leur permettent ou non d'accéder à leur profil. Ces dits profils serait reliées à une base de donnée (une mongoDB), qui contiendrait les informations importantes, comme le nom, les

préférences, etc. Cette base de donnée devrait communiquer avec les autres bases de données reliées aux données générées par les utilisateurs.

En somme, l'ajout d'utilisateur demanderait non seulement un frontend plaisant à l'oeil, mais qui sert aussi de couche applicative pour relier les informations à des utilisateurs (et restaurant) entre les divers BD.

5.1.2 Nouvelles données - liées aux utilisateurs

Une BD elasticsearch devrait tenir les tous les reviews des utilisateurs. Il faudrait donc des identifiants savoir quel utilisateur a fait quel review, mais aussi de mettre un identifiant de restaurant dans le review. Cet identifiant de restaurant permettrait de ne plus conserver les reviews dans les les nodes Restaurants de Neo4j, mais aussi de faire les calculs comme la moyenne des reviews dynamiquement, à chaque *upsert* de reviews.

De plus, l'ajout d'une BD Elasticsearch permettrait permettrait de soutenir des ajouts et mises à jour de reviews facilement, en plus de stocker toutes les données textuelles et d'effectuer des recherches dans le texte, source de nouvelles fonctionnalitées.

Avec l'état actuel de l'application, les reviews peuvent être ajoutés dans les nodes Restaurants, mais ceci causerait des problèmes pour les valeurs des moyennes… Tout comme effacer et modifier les reviews, car il faut repasser par le script de traitement de la langue (comprend un ETL et un modèle, voir section 6). Avoir des reviews dans ElasticSearch permettrait aussi de faire d'ajouter des champs intéressant supplémentaires dans les reviews. Non pas seulement avoir des cotes étoilées globales, mais aussi pour le service, la présentation, l'expérience, l'ambiance, la nourriture, la propreté, le temps de service, le temps de commande, etc. La plupart de ces données peuvent être reliés au type du restaurant et à l'achalandage! (Possibilité de faire un graphe en étoile sur le type d'un resto! Ex: 40% fastfood, 100% Buffet, 20% Pub et 0% 5 étoiles!).

Les utilisateurs pourraient aussi émettre leur propre parcours (à la main), noter et reviews des parcours, qui peuvent ensuite être suggérés et partagés à d'autres utilisateurs! Un autre index dans la BD ElasticSearch pourrait contenir l'information des review de parcours et offrir les même fonctionnalitées que les reviews de restaurants.

L'utilisation d'un GPS lié à l'utilisation de l'application pourrait fournir des données sur les temps de déplacements et les temps à chaque restaurant. Ceci pourrait servir de statistiques pour vérifier et améliorer les algorithmes d'estimation du trafic (nouvelle fonctionnalité liées à des données externes aux utilisateurs) et du temps de déplacement. De plus, les données permettraient d'amasser des statistiques lors des parcours. Ces statistiques, par parcours ou globale, pourraient elles aussi être partagées ou non.

5.2 Gestion de données additionnelles

5.2.1 Nouvelles données

Avec des données plus fiables sur les types de restaurants (buffet, pub, fastfood, etc.) et les chaînes de restaurant, les nodes Restaurants de la Neo4j pourraient avec plus d'un type de Node (Restaurant, Subway, Buffet) à la fois et profiter de l'indexation à la recherche, accélérant les queries. Ce genre de valeur ne devrait pas être changé par les utilisateurs, mais les gestionnaires de BD (via les entreprises). Le tout devrait avoir peu d'impact sur la Neo4j à cause du faible volume d'écriture, pour un grand impact sur la rapidité en lecture pour plusieurs recherches!

Il serait pertinent d'obtenir des informations supplémentaires comme le trafic sur le réseau routier puisqu'un cycliste a le droit de circuler sur les routes standard. Ceci demanderait de retravailler le graphe des routes pour inclure de nouveaux type de node. Les nodes Intersection pourraient être séparés en type "Cyclable" et type "Routier", les indexer, et avoir des nodes qui sont des types à la fois. Les parcours pourraient donc être choisis en fonction du type de route.

Dans certain cas, les pistes cyclables sont disjointes, il serait donc possible de trouver des “connexions” qui permettent à un utilisateur de passer par une route pour rejoindre une autre piste cyclable qui commence un peu plus loin.
La partie GPS de l’application permettrait de fournir des informations météo sur le temps présent et prévue pour une date de trajet. Ces informations sur la température, pourrait provenir d’un api externe vers un système et ne pas être stockée directement dans aucune base de données, mais seulement utilisée “*live*”. Elles pourraient aussi être stockées aux parcours directement et être utilisées comme variables exogènes pour, par exemple, normaliser les cotes des routes.

Il serait intéressant d’aller chercher des données complémentaires sur les restaurants, tel que les heures d’ouverture, les menus et les événement de la ville, afin d’offrir plus de critères de sélections pour les épicuriens et les tris de trajets.

Obtenir l’achalandage, ainsi que les heures horaires des restaurants grâce à l’information que possède Google. Obtenir de l’information sur les grands évènements de la ville en passant par quebecoriginal.com. Il serait aisé de mettre les menu pour tous restaurants possédants un site web simplement en créant un lien vers celui-ci.

La plupart des données temporelles pour la météo et l’achalandage pourraient être placées dans une InfluxDB, permettant d’avoir une granularité temporelle intéressante, un grand historique, des agrégations rapides et une recherche par plage de temps!

5.2.2 Gestion globale des données et expansion

Ajouter autant de données, dans plusieurs nouvelles BD, avec plusieurs nouvelles collection amène non seulement du temps de développement, mais aussi des questions sur la réplication des données et partitionnement lors du déploiement. Les réponses vont nécessairement varier avec le cas d’utilisation.

Au départ, les données proviennent majoritairement de sources externes et sont précompilées. En cas de problème, les BD peuvent être *seeded* à neuf en moins de 30 minutes, dans le cas

où tout est perdu. Peu d'utilisateur et aucune réputation implique aussi un *load* faible et probablement que d'avoir des pannes de 30 minute ne serait pas un grand problème.

Toutefois, à mesure que les données propriétaires s'ajouteront, la perte des données deviendrait de plus en plus problématique, car ce genre de données est une mine d'or et qu'une perte, même à 10%, peut être une perte considérable.

Plus le service prendra forme et que les utilisateurs utilisent l'application, plus le service ne pourra pas tomber en panne sans créer de vague de mécontent… surtout dans l'ère du cloud et du "*available-at-all-times*". D'autres services plus fiables prendront la place rapidement.

Il faudrait donc définitivement répliquer et distribuer les bases de données sur plusieurs machines, qui devraient être suffisamment étendues pour que le service survive à des pannes localisées géographiquement (dépend des régions où l'on veut déployer l'application). La taille de cet étendu dépendra, dans les fait, de la répartition géographique des routes et des restaurants.

Heureusement, la plupart des données prévues pour être amassées se font via les utilisateurs où des sources externes, ce qui permet d'aller étendre le système partout où des routes et pistes cyclable (et des restaurants!) existent. Mongo, Neo, Elastic et même Influx sont toutes des BD pouvant être liées aux services cloud comme Amazon Web Services ou Microsoft Azure, offrant une grande flexibilité pour la répartition géographique. Étant un facilitateur pour les restaurants, ces derniers pourraient aussi facilement aider à fournir leurs informations dans l'application gratuitement.

5.3 Fonctionnalitées supplémentaires

- Permettre aux utilisateurs d'entrer leur propres reviews
    - Un nouveau endpoint serait ajouté pour permettre aux usagers d'envoyer leur propres reviews contenant text et/ou une cote, 1 à 5 étoile, qu'il serait ensuite possible de traduire par notre système en une probabilité d'être positif.

        - Pour des utilisateurs “étranger” ou à l'étranger, les menus (et les reviews) pourraient être traduits!
- Permettre aux utilisateurs de faire des reviews autres que global (pour le service, pour la nourriture, pour le décor, l’expérience, l’accessibilité, les prix, etc.) et de noter les éléments différemment
    - Lors de l’envoie de review, un utilisateur pourrait choisir parmis un ensemble de type de review ce qui donnerait des statistiques plus près des faits pour chaque catégorie. Il serait possible d’envoyer une review de chaque type.
- Permettre aux utilisateurs de faire de la recherche de terme pour les reviews
    - En passant par ElasticSearch pour les reviews, il deviendrait aisé de faire une recherche à l’intérieur des reviews pour trouver certains termes qui correspondent à ce que l’on désire, il serait aussi possible de chercher les synonymes.
- Intégrer les menus
    - Permet aux utilisateurs d’obtenir un trajet selon le type de nourriture qu’ils ont le goût de manger. Une technique pour ce faire serait d’utiliser une liste de type de repas des restaurants et de les agréger aux nodes de restaurant et la requête appliquerait le même principe que pour le les types de restaurants.
    - Avec suffisamment de menu (en image) Il serait possible de développer un réseau de neurone qui extrait des photos des menus, les items.
        - Ceci pourrait se faire via une fonction “photo du menu” de l’application
        - Pour des utilisateurs “étranger” ou à l'étranger, les menus (et les reviews) pourraient être traduits!
- Mettre en garde l’utilisateur s’il annonce du mauvais temps dans sa région.
    - Grâce à un api qui fournit les informations météorologiques, il serait possible d’avertir le client, avant qu’il commence, de la possibilité de mauvais temps et ainsi assurer que l’usager ne se fasse pas prendre au dépourvu pendant son trajet, assurant une sortie aussi agréable que possible.
- Offrir un trajet qui tient en compte les heures d’ouverture des restaurants.
    - Utilise les horaires de ceux-ci, une liste de liste contenant les heures de début et de fin selon le jour de la semaine, pour ajouter un filtre dans les requêtes de parcours afin de filtrer les trajets proposés à l’utilisateur pour ne prendre en

considération que ceux qui sont ouverts, ou ceux qui seraient ouvert selon les estimations de temps de trajet (et de temps à chaque stop!)

- Permettre aux utilisateurs de choisir des parcours similaires à ceux qu'ils ont aimés.
    - Utilise un historique des parcours appréciés de l'utilisateurs pour soumettre des parcours similaire en longueur et en type de restaurant. Il suffirait de renvoyer un payload dont les informations sont les mêmes à l'exception du point de départ qui se lancerait en arrière plan dès qu'un utilisateur indique qu'il a aimé son parcours, de manière à remplir une liste de recommandation future.
- Gérer les sens des pistes cyclables.
    - Les données de la ville de québec fournisse déjà la direction de la piste cyclable, un travail plus avancé donnant de l'information plus précise pour le client en liant dans la Neo4j en respectant le sens de ceux-ci
- Offrir de faire une recherche de trajet pour une certaine durée plutôt qu'une certaine longueur.
    - En recueillant les informations des utilisateurs lors de leur trajet il serait possible d'obtenir un temps pour chaque segment, qui contrairement à la distance pourrait varier selon la direction puisqu'il est bien plus rapide de descendre une côte que de la monter.
- Envoyer des défis à ses amis sur l'application.
    - Avec les informations sur le temps pris pour un certain trajet, on pourrait ainsi créer une saine compétition (vitesse à vélo, pas à manger) en permettant d'envoyer ces données à ses amis. Il serait aussi intéressant d'utiliser cette fonctionnalité pour acquérir de nouveau utilisateur en permettant l'envoie de ces défis et des gens qui n'ont pas l'application en passant plutôt par un numéro de téléphone, un texto, une adresse courriel ou un post sur les médias sociaux!
        - *"Speedy-Foodie"* [2].

[2] (Trademark pending) Vraiment en train de faire des recherches pour le trademark

6. Fonctionnalités avancées

Cette section détaille l'origine des fonctionnalitées avancées et additionnelles présentées dans les sections précédentes.

6.1 Fonctionnalités

L'application offre actuellement, comme présenté plus haut, un endpoint de parcours du critique, faisant usage d'un indice sur la qualité des restaurants, ainsi qu'un support pour obtenir des reviews par nom de restaurant (ex: tout les Yuzu Sushi) et par restaurant en particulier (ex: Seulement le Yuzu sur Charest Est.).

L'application offre aussi des fonctions de calculs pour la distance sur un ellipse, trouver des intersections sur des plans elliptiques, des *bearings* (direction géospatiales), qui sont particulièrement utiles pour gérer des graphs, mais aussi dans des fonctionnalitées futures, comme déterminer l'orientation d'un trajet seulement avec des coordonnées géospatiales à travers le temps.

L'indice, soit la probabilité des critiques, est un champs créé en utilisant des techniques d'apprentissage automatique et de traitement de la langue naturel (NLP). Pour obtenir de telles probabilités, un processus d'extraction/transformation/chargement (extract/transform/load; ETL), ainsi qu'un modèle sous forme de régression doivent être mis en place. Les entrées du processus ETL doivent être du texte de reviews et retourner une paire de "features - target" que le modèle pourra utiliser. La cible, dans ce cas, sont les notes de 1 à 5 étoiles (entiers) allant avec chacun des reviews.

6.2 Modèles

Plusieurs modèles peuvent répondre à ces questions, des modèles de machine learning classiques au réseau profond, en passant par l'apprentissage par renforcement. Dans le cas présent, un random forest a été choisi, principalement pour son explicabilité, sa rapidité d'entraînement dans un cas NLP et sa rapidité d'exécution. Ces critères sont utiles pour ré-entraîner rapidement, comprendre les points majeurs des choix du modèle (pour générer

plus de valeurs avec les données, potentiellement en offrant des fonctionnalitées additionnelles dans le futur) et aussi pour s'exécuter en parallèle sur de petite machine, ce que plusieurs modèle de deep learning pourrait avoir de la difficulté à faire, advenant un très grand volume de nouveaux reviews simultanés, ou un faible budget hardware. Il ne serait toutefois pas impossible d'utiliser plusieurs modèles et d'avoir un indice par modèle, pour améliorer les prédictions.

6.3 ETL - De l'entrée du processus à des "features" pour l'entraînement

- Extract
    - Les reviews sont extrait directement, pour chaque restaurant, un à un, avec la note associée
- Transform
    - Les reviews sont filtrés pour retirer des caractères spéciaux
    - Ils sont ensuite séparé en jeton, selon leur langue
    - Les *stopwords* et la ponctuation sont retirés
    - Les jetons sont réduit à leur racine (*stemming*)
    - Le sentiment de chaque jeton-racine est extrait
        - La transformation est fait à l'aide de la librairie python NLTK, par des scripts de preprocessing. Il est à noter que cette partie pourrait aussi être automatisée directement dans ElasticSearch, point important si les reviews peuvent être modifiées ou ajoutées dynamiquement dans l'application, au lieu d'updates par batch lors de mises-à-jours du système.
- Load
    - Les sentiments et les cotes sont chargées

6.4 Entraînement

Utilisant les sorties du processus ETL, les paires critiques/notes ont été divisées en groupe d'entraînement et de test, 4000 pour le premier et 1000 pour le deuxième. Les critiques du groupe de test, ainsi que leur probabilité d'être une critique positive ont été ajouté aux

restaurants en répartissant de la manière la plus uniforme possible, afin d’avoir au moins 2 reviews sur tous les restaurants.