19/01/2020

MAC - SyugarDaddyBot

Loris Crüll, Julien Rod, Yoann Rohrbasser

# 1 Cahier des charges

## Description

Notre projet consiste en un bot disponible sur l’application de messagerie Telegram permettant à tout utilisateur de lui envoyer diverses commandes qu’il se chargera d’exécuter. Ces commandes concernent principalement la création ainsi que la consultation de recettes de cuisine partagées par d’autres utilisateurs. De plus, ils ont la possibilité de montrer qu’ils ont apprécié une recette, un ustensile ou encore un autre utilisateur via des « Likes ».

## Fonctionnalités

Ci-dessous la liste des commandes pouvant être entrées par un utilisateur lorsqu’il s’adresse au bot :

|  |  |
| --- | --- |
| Commande | Description |
| /newrecipe [nom] | Démarre la création de la recette [nom] |
| /reset | Met fin au processus de création d'une recette |
| /random | Affiche une recette aléatoire |
| /getrecipe [id] | Affiche la recette ayant pour id [id] |
| /recipesbyingredients [i1, i2, ...] | Affiche les recettes ayant pour ingrédients [i1, i2, ...] |
| /recipesbyuser [utilisateur] | Affiche les recettes de l'utilisateur [utilisateur] |
| /recipesbycalories [calories] | Affiche les recettes ayant moins de [calories] calories |
| /recipesbytools [t1, t2, ...] | Affiche les recettes ayant pour ustensiles [t1, t2, ...] |
| /recipesbytime [temps] | Affiche les recettes prenant [temps] à réaliser à 5 minutes près |
| /userscooking [id] | Affiche les utilisateurs ayant fait des recettes similaires |
| /recommandations | Propose des recettes sur la base de celles consultées jusqu'à présent |
| /help | Affiche la liste des commandes disponibles |

# 2 Modèle des données

## 2.1 Bases de données

Nous avons choisi MongoDB comme base de données orientée document. L’avantage d’une telle structure est qu’il n’est plus nécessaire de faire des jointures pour reconstituer l'information car elle n'est plus dispersée dans plusieurs tables. Contrairement au SQL, le NoSQL permet la gestion de données flexibles rendant la tâche simple lors de l’écriture des requêtes. Une lecture unique suffit à reconstituer les documents et ceux-ci n’étant pas dépendants les uns des autres, sont facilement déplaçables. La représentation des données permet une sérialisation aisée peu importe la complexité de sa structure. Elle nous permet principalement de stocker et retrouver les données sans se soucier des relations les liant.

Nous avons choisi Neo4j comme base de données orientée graphe. En plus de la curiosité de découvrir le fonctionnement et la mise en pratique de ce type de base de données, Neo4j répondait parfaitement aux attentes de notre application : données connectées, dynamiques et requêtes complexes en temps réel (likes, utilisateurs similaires et recommandations). Elle nous permet d’établir facilement les liens entre les données nous permettant ainsi de les retrouver tout en appliquant plusieurs filtres en fonction des liens.

## 2.2 Source de données

Plusieurs acteurs entrent en compte dans notre application. Les utilisateurs peuvent consulter ou créer de nouvelles recettes, elles-mêmes composées d’ingrédients et nécessitant des ustensiles précis. L’appui sur le bouton « Like » permet à l’utilisateur de montrer qu’il apprécie une certaine recette, un certain ingrédient, un certain ustensile ou encore un certain utilisateur.

### 2.2.1 Collections

Utilisateur : comporte l’id de l’utilisateur.  
Ingrédient : comporte le nom de l’ingrédient.  
Ustensile : comporte le nom de l’ustensile.  
Recette : comporte le nom de la recette, sa description, son temps de préparation et son nombre de calories. Il est possible d’ajouter des tags supplémentaires à cette collection pour préciser une sous-catégorie (gâteau, pâtisserie japonaise, glace, etc…).

### 2.2.2 Relations

IN : Part d’un ingrédient et va sur une recette. Indique quel ingrédient utiliser pour suivre la recette. Elle comprend la quantité de l’ingrédient à utiliser.  
USEFULL : Part d’un ustensile et va sur recette. Indique quel ustensile utiliser pour suivre la recette.  
PROPOSED : Part d’un utilisateur et va sur une recette. Indique l’utilisateur qui a proposé la recette. Elle comprend la date de la proposition.  
LIKE : Part d’un utilisateur et va sur un objet de n’importe quelle collection. Indique que l’utilisateur a aimé l’objet en question. Elle comprend l’heure de la relation établie.

Ci-dessous quelques extraits du fichier DataModelAndAnalyticRequest.txt :  
CREATE (Sucre:Ingredient {name:'sucre'})  
CREATE (User5:User {name:'\_4'})  
CREATE (Casserole:Ustensile {name:'casserole'})  
CREATE (PommeDAmour:Recipe:Fruit {name: '\_5e21c08cadc3434914bc7d88'})  
CREATE (Sucre)-[:IN{quantite:'70g'}]->(PommeDAmour)  
CREATE (Casserole)-[:USEFULL]->(PommeDAmour)  
CREATE (User5)-[:PROPOSED]->(PommeDAmour)  
CREATE (User1)-[:LIKE]->(FarineDeRiz)  
CREATE (User5)-[:LIKE]->(Ecumoir)  
CREATE (User7)-[:LIKE]->(User5)

Une image contenant capture d’écran

Description générée automatiquement

### 2.2.3 Données

Les fichiers user.csv et recette.csv servent à remplir la base de données MongoDB, mais il est nécessaire de changer le type des champs kcal, time et id en int32 après l'importation, puis d’effectuer la correspondance entre les objectsId des recettes de MongoDB avant d’insérer les données du fichier DataModelAndAnalyticRequest.txt dans Neo4j. Ce remplissage sert à tester notre bot.

# 3 Implémentation

## 3.1 Environnement

Pour mettre en place notre environnement, nous avons installé MongoDB, créé la base de données SyugarDaddy puis les tables User et Recipe avant d’y importer les données des fichiers csv une fois les manipulations nécessaires effectuées. Concernant Neo4j, une fois la base de données créée et démarrée, il suffisait d’importer le contenu du fichier DataModelAndAnalyticRequest.txt (après avoir fait la correspondance avec les ids des recette dans MongoDB). Une fois ceci fait, le bot peut être lancé et il est ainsi possible de lui écrire des commandes via Telegram.

## 3.2 Structure du code

### 3.2.1 Main.java

Cette classe se charge simplement d’initialiser le bot et de le lancer. À noter qu’il lui est nécessaire d’avoir un nom disponible et un token valide.

### 3.2.2 Bot.java

Cette classe se charge de traiter les commandes entrées par l’utilisateur. Lors du premier message reçu, le bot va créer le profil de l’utilisateur puis l’insérer dans MangoDB et créer son nœud dans Neo4j. Si l’utilisateur ne respecte pas le format d’une commande, il en est par la suite informé. Dans le cas où la fonctionnalité « Like » est disponible à la suite d’une commande, le bouton correspondant s’affiche en-dessous.

### 3.2.3 MongoDBDAO.java

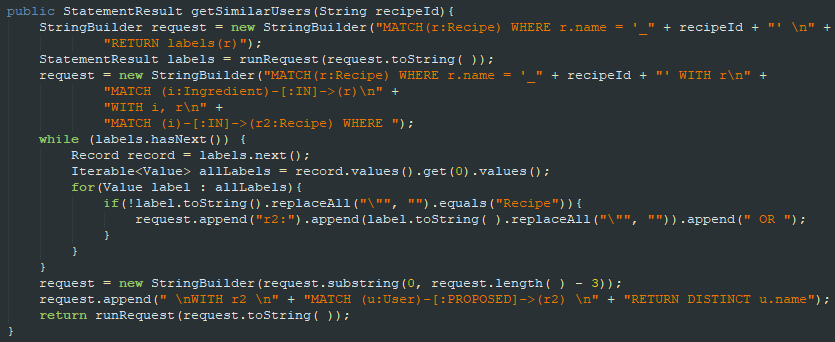
Cette classe se charge de gérer les actions en lien avec MongoDB. Elle peut vérifier l’existence d’un utilisateur, ajouter une recette, retourner une recette aléatoire, trouver une recette selon son id, trouver un utilisateur selon son id et trouver une recette selon son temps de préparation ou ses calories. Les requêtes sont simples et n’impliquent qu’une seule table.

### 3.2.4 Neo4jDAO.java

Cette classe se charge de gérer les actions en lien avec Neo4j. Elle peut établir les liens entre les diverses entités en jeu lors de l’ajout d’une recette ou d’un like et effectuer des requêtes complexes impliquant plusieurs niveaux de relations entre les nœuds.

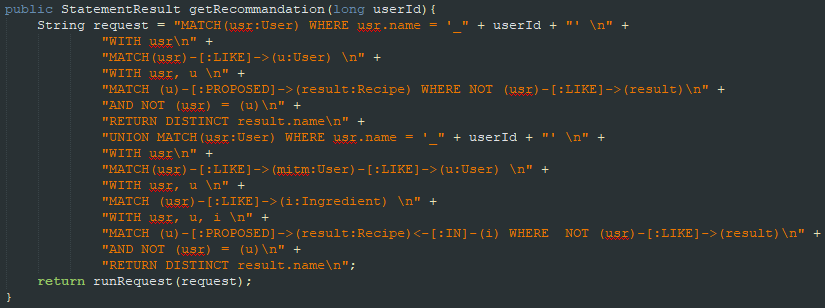
## 3.3 Requêtes avancées

### 3.3.1 /userscooking



Cette commande retourne la liste des utilisateurs ayant proposé des recettes similaires à celle entrée par l’utilisateur. On se base sur les ingrédients utilisés par la recette entrée et ses sous-catégories afin que le résultat obtenu soit le plus pertinent possible pour l’utilisateur.

### 3.3.2 /recommandations



Cette commande retourne une liste de recettes recommandées sur la base des likes de l’utilisateur. Plus précisément, on regarde les recettes qui n’ont pas encore été likées d’un utilisateur lui-même liké par l’utilisateur courant, ainsi que les recettes d’utilisateurs likés par l’utilisateur liké ayant au moins un ingrédient liké par l’utilisateur courant.