

# HUNGRY.ME

Projet de MAC

Edouard de Chambrier, Adrian Mayo Cartes

HEIG-VD

19.01.2020

# 1. Cahier des charges

## 1.1 Objectifs

Le but de ce projet est de créer un logiciel qui gère une base de données orientée document et une orientée graphe. La communication entre les bases de données et l'utilisateur se fait grâce à une API tierce.

## 1.2 Technologies

Plus précisément, notre projet consiste à créer une application de gestions de recettes baptisé **HUNGRY.ME**. Ce projet utilise **MongoDB** comme base de données orientée document, **Neo4j** comme base de données orientée graphe et Telegram comme API tierce faisant le lien homme-machine.

Pour faciliter l'implémentation des différentes technologies, le langage de programmation *Java* est utilisé ainsi que l'infrastructure *Maven* pour la gestion des différents modules.

## 1.3 Fonctionnalités

Une fois connecté à **@HungryMe\_bot** via Telegram, l'utilisateur doit être capable de demander des recettes suivant ce format: (a) {time : (very) fast/short} recipe(s) with {ingredients} for {tags}.

Les recettes retourné par l'API doivent pouvoir être évaluées soit en l'approuvant ("pouce vers le haut" 👍), soit en la désapprouvant ("pouce vers le bas" 👎), soit en l'ajoutant à ses favoris ("cœur" ❤️) ou, s'il est déjà dans les favoris, en la retirant ("cœur brisé" 💔).

Les recettes retourné à l'utilisateur doivent être pertinentes; dans ce cas, l'application devra retourner des recettes soit proche en terme d'ingrédients ou de tags (si non spécifié dans la requête) par rapport à ceux qu'il a déjà apprécié ou mis en favoris ou soit, s'il n'a pas de favoris ou d'appréciée, les plus appréciées par les utilisateurs.

## 1.4 Projection

A terme, il devrait être possible pour l'utilisateur d'ajouté de nouvelles recettes, de créer des requêtes plus complexes (promouvoir ou exclure des ingrédients), d'avoir des conversion d'unités (1 cup = 8.1 oz = 240ml) ou encore de se baser sur une recette et de la modifier à ça préférence. Cependant, ces points ne seront pas dans ce projet.

## 2. Modèle des données

Le projet utilise deux types de base de données: une base de données orientée document et une orientée graphe. Dans le cadre de ce projet, la base de données orientée document est utilisée pour stocker les différentes recettes avec tous les détails associées (nom, ingrédients, tags, calories, protéines, etc..). Quand à la base de données orientée graphe, elle est utilisée pour stocker et rechercher les différentes relations entre les recettes, ingrédients et tags.

Les données des recettes utilisées proviennent ce répertoire GitHub: <https://github.com/tabatkins/recipe-db>.

### 2.1 Modèle document

La base de données orientée document utilisée est **MongoDB**. Par rapport au fichier d'origine recipe-db.json, les données doivent être traitées pour les uniformiser à l'utilisation du logiciel. Un objet, correspondant à une recette, utilise le format de l'exemple suivant:

```
1      {
2          "_id": ObjectID("5e1c684c88edd028a6bd546b"),
3          "name": "Black Forest Bircher",
4          "source": "https://www.bbcgoodfood.com/recipes/black-forest-bircher",
5          "preptime": 600,
6          "waittime": 0,
7          "cooktime": 0,
8          "servings": 2,
9          "comments": "",
10         "calories": 536,
11         "fat": 46,
12         "satfat": 10,
13         "carbs": 222,
14         "fiber": 10,
15         "sugar": 56,
16         "protein": 22,
17         "instructions": "Combine the pears, oats, ...",
18         "ingredients": [
19             {
20                 "unit": "unit",
21                 "quantity": 1,
22                 "name": "small pear, grated\r"
23             },
24             {
25                 "unit": "cup",
26                 "quantity": 0.25,
27                 "name": "rolled oats\r"
28             },
29             {
30                 "unit": "spoon",
31                 "quantity": 0.5,
32                 "name": "unsweetened cocoa powder\r"
33             },
34             {
35                 "unit": "cup",
36                 "quantity": 0.25,
37                 "name": "Greek yogurt\r"
38             },
39         ]
40     }
```

```

39         {
40             "unit": "spoon",
41             "quantity": 2,
42             "name": "milk\r"
43         },
44         {
45             "unit": "spoon",
46             "quantity": 0.5,
47             "name": "honey, plus extra to serve (optional)\r"
48         },
49         {
50             "unit": "cup",
51             "quantity": 0.5,
52             "name": "cherries, halved and pitted\r"
53         },
54         {
55             "unit": "spoon",
56             "quantity": 2,
57             "name": "Greek yogurt\r"
58         },
59         {
60             "unit": "oz",
61             "quantity": 1,
62             "name": "semisweet chocolate chips"
63         }
64     ],
65     "tags": [
66         "untried",
67         "vegetarian",
68         "breakfast",
69         "quick",
70         "make_ahead"
71     ]
72 }

```

Concernant le champs `_id`, il correspond à la méthode utilisée par **MongoDB** pour stocker les informations. Les ingrédients utilise le format impérial en place et lieu du système métrique. A terme, le logiciel devrait être capable de convertir d'un système à un autre ou de mélanger les deux cependant il n'est pas implémenter dans cette version.

## 2.2 Modèle graphe

La base de données orientée graphe utilisée est **Neo4j**. Les données utilisées proviennent de la base de données orientée document **MongoDB**.

Les différents types de nœud utilisés sont :

- **Recipe** : contenant l'id "type MongoDB", le nom ainsi que le temps pour cuisiner, de préparation et d'attente.
- **Ingredient** : contenant le nom de l'ingrédient.
- **Tag** : contenant le nom du tag.
- **User** : contenant l'id "type Telegram", le prénom le nom et les préférences de temps (la valeur de `fast very fast`).

ainsi que les relations :

- **contains** : relation de Recipe vers Ingredient.
- **has** : relation de Recipe vers Tag.
- **looked** : relation de User vers Recipe. Cette relation contient les propriétés **liked**, **disliked** et **favorite**. Cela permet de faciliter la création et recherche des liens et de savoir en plus que quelles recettes un utilisateur a consulté.

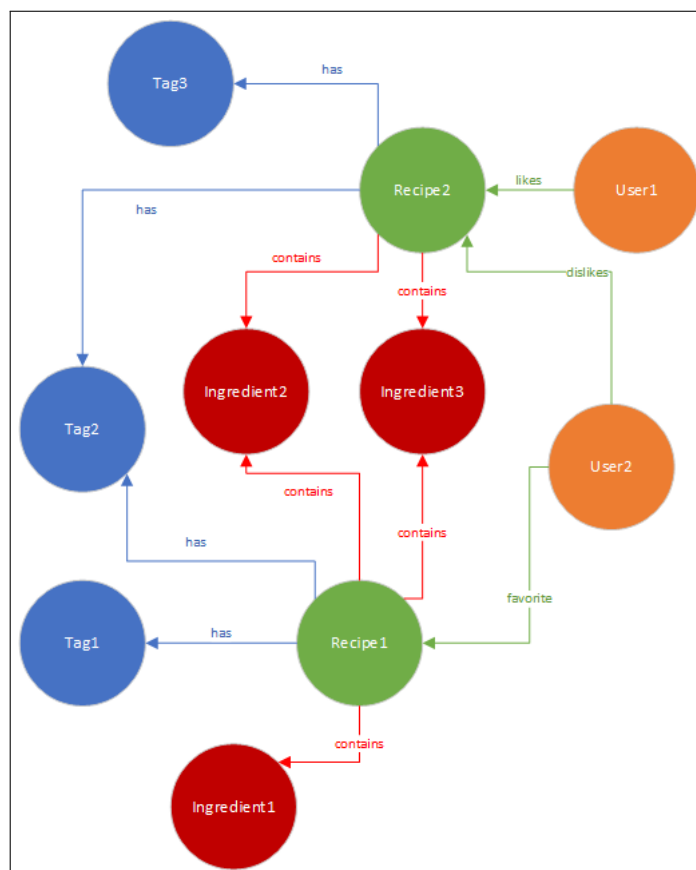


Figure 2.1: Représentation du graphe

## 3. Requêtes


### 3.1 Requêtes standard

Il existe plusieurs requêtes existantes sur l'application qui ne sont pas considérées comme avancées mais sont utiles au bon fonctionnement de l'application.

les requêtes sont les suivantes :





1. La recherche d'une ou plusieurs recettes peut se faire grâce à la requête :

(a) `{time : (very) fast/short} recipe(s) with {ingredients} for {tags}`

2. L'accès aux favoris peut se faire grâce aux commandes : `liked`, `favs`, `favorites` ou encore le symbole cœur .

### 3.2 Requêtes avancées

La requête spéciale que l'utilisateur peut effectuer sert à lui proposer des recettes de la manière suivante :

1. Les recettes retournées seront des recettes que l'utilisateur n'aura pas encore vues (donc avec lesquelles il n'a pas de lien) qui ont un maximum d'ingrédients semblables aux recettes que l'utilisateur a appréciées () ou mises en favoris (.
2. Si l'utilisateur n'a pas encore appréciée () ou mise en favoris () des recettes, une liste de recettes les plus appréciées et mises en favoris par l'ensemble des utilisateurs lui sera proposée. Les recettes proposées seront également filtrées afin de n'afficher que des recettes que l'utilisateur n'a pas encore vues (c'est l'action de cliquer sur une recette).
3. Dans le (peu probable) cas où l'utilisateur aurait vu toutes les recettes, les mêmes recettes que le point précédent (mais cette fois non filtrées) lui seront proposées.

Afin de lancer la requête, l'utilisateur doit envoyer le message `~\_ (ツ) _/~` (accessible sur Windows grâce au raccourci Win+.) ou `surprise` au bot HungryMeBot.

## 4. Structure code

### 4.1 Structure globale

Le projet est subdivisé en plusieurs modules indépendant contenant les logiques des différentes technologies utilisées. Par exemple, le module `telegram` ne sait pas que **MongoDB** et **Neo4j** sont utilisés, il ne connaît que les interfaces `DocumentDatabase` et `GraphDatabase`. Cela permet une grande modularité pour les technologies utilisées, dans l'éventualité où l'on souhaiterait changer ces dernières.

```
ch.heigvd.mac.hungryme
|
+-- interfaces
|
+-- models
|
+-- mongodb
|
+-- neo4j
|
+-- telegram
|
+-- Env.java
|
+-- Installer.java
|
+-- Main.java
|
+-- Utils.java
```

Les interfaces et modèles sont les modules contenant les classes utilisées pour la communication inter-modules.

Les classes `Installer` et `Main` permettent respectivement d'installer les données dans les bases de données et de lancer le logiciel principal. `Env`, quand à lui, contient les constantes utilisées pour accéder aux bases de données ou au bot de Telegram.

### 4.2 Module interfaces

Le module `interfaces` contient les interfaces implémentées par les différents contrôleurs. Par exemple, `DocumentDatabase` est implémenter par le contrôleur du module de **MongoDB**. Ces interfaces permettent de gérer les point d'entrées des différent modules et ce que que ces différents modules pourront faire.

```
ch.heigvd.mac.hungryme.interfaces
|
+-- DocumentDatabase.java
|
+-- GraphDatabase.java
```

## 4.3 Module models

Le module `models` contient les différents types d'objet que les modules de l'application peuvent se transférer. Cela permet d'avoir la même logique comprise par tous le monde sans avoir à formater en recevant les données.

```
ch.heigvd.mac.hungryme.models
|
+-- Ingredient.java
|
+-- Recipe.java
|
+-- Unit.java
|
+-- User.java
```

## 4.4 Module mongodb

Le module `mongodb` contient les éléments utilisé pour accéder à la base de données **MongoDB**.

La classe `MongoDBController` contient toute la logique qui permet de se connecter à la base de données, envoyer et récupérer les informations. Le sous-module `parsers` contient toute la logique qui permet de récupérer les données du fichier d'origine `recipe-db.json`, de les traiter pour uniformiser les données, par exemple "1 1/2 lb wheat" est converti en "1.5 pound wheat", et finalement de les transformer en format utilisable par **MongoDB**.

```
ch.heigvd.mac.hungryme.mongodb
|
+-- parsers
|   |
|   +-- JSONParser.java
|   |
|   +-- MongoDBParser.java
|   |
|   +-- ParserBuilder.java
|
+-- MongoDBController.java
```

## 4.5 Module neo4j

Le module `neo4j` contient les éléments utilisé pour accéder à la base de données **Neo4j**.

La classe `Neo4jController` contient la logique qui permet d'accéder à la base de données, envoyer et récupérer les nœuds et relations et qui permettra de chercher les recommandations selon certains critères prédéfinis.

```
ch.heigvd.mac.hungryme.neo4j
|
+-- Neo4jController.java
```

## 4.6 Module telegram

Le module `telegram` contient l'API qui permet la communication entre les bases de données et les utilisateurs.



Il s'agit d'un bot, du nom de HungryMeBot, qui permet de récupérer les demandes des utilisateurs. Il retourne à ces derniers les informations détaillées des recettes ainsi que les avis, sous forme d'emojis ("pouce en haut", "pouce en bas", "cœur" et "cœur brisé")

```
ch.heigvd.mac.hungryme.telegram
|
+-- HungryMeBot.java
```

## 5. Problèmes rencontrés

### 5.1 Traitement des données

L'un des problèmes rencontré fut la conversion du fichier de données original en données exploitable par **MongoDB** surtout pour les ingrédients. Le problème est que, outre le fait que le système impérial soit utilisé, est que les valeurs soient en fractions, par exemple "3/4" qui fait 0.75 ou "1 1/2" qui fait 1.5.

Certaines simplifications ont dû être apportées comme les "teaspoon" ou "tablespoon" ont dû être simplifiées en "spoon". Aussi, les ingrédients en eux-mêmes sont parsés très simplement. Par exemple, il n'y a pas de simplification si l'on veut du "pepper", on ne retournera pas les ingrédients comme "red pepper" ou "black pepper".

### 5.2 Java & Neo4j

Concernant l'API Java, lors de l'import des données depuis **MongoDB**, il n'était pas possible de créer les nœuds et les liens lors de la même session; cela créait des relations disjointes.

Il était nécessaire d'ouvrir une session avec un `try`, créer tous les nœuds, fermer le `try` puis en réouvrant un autre pour faire les relations.

### 5.3 Limitation du nombre de recettes

Pour que l'utilisateur puisse choisir la recette qu'il souhaite voir parmi la liste retournée, c'est des boutons qui sont renvoyés en message de retour. Ces boutons prennent une chaîne de caractères en callback. Par exemple l'id de la recette si on appuie sur le nom d'une recette, le symbole suivie de l'id de la recette dans le cas du like ou encore les ids des recettes suivantes séparés par un espace dans le cas "**Show more**".

Le problème est que la longueur de la chaîne est limitée donc il n'est possible de mettre que l'id de deux recettes au lieu de tous les autres restant.