Eveille tes papilles

Table des matières

Contexte et objectifs du projet

Dans le cadre du cours Web Mining, ci-après WEM, nous devons réaliser un projet nous permettant de mettre en pratique les connaissances acquises. Le WEM désigne l'ensemble des techniques visant à explorer, traiter et analyser les masses d'informations liées à une activité web.

Le WEM permet, à partir de données mises à disposition sur le web, d'ajouter de la valeur et d'en tirer profit. L'application « Eveille tes papilles » servira à guider les utilisateurs sur leur choix de recettes culinaire, tout en leur proposant de nouvelles recettes ou des recettes adaptées aux ingrédients à leur dispositions. Il existe déjà différents sites internet utilisant ce principe, par exemple « Swissmilk » et sa section recettes-idées : https://www.swissmilk.ch/fr/recettes-idees/.
Le petit plus qui fera toute la différence c'est que notre application sera complétement orienté utilisateur et lui proposera des recettes, sans qu'il ne doive passer trop temps à faire des recherches et les trier.

De plus, une fonction innovante, « Wine pairing », réjouira l'utilisateur en lui proposant le vin le mieux adapter à son menu.

Données

Sources

L'acquisition des données sera réalisée en crawlant les pages web avec un robot « Spider Scrapy ». Cela donnera des fichiers au format « JSON » contenant les infos souhaitées (voir section ci-dessous).
Les données des recettes de cuisine seront acquises depuis le site https://www.Marmiton.org. Les recettes incluses dans le scope du projet sont : entrée, repas et dessert.

Les données concernant les vins seront acquises depuis le site https://www.Coop.ch/fr/vins/c/m_0222. Seuls les vins rouges et vins blancs seront utilisés.

Il avait également été envisagé d'utiliser les données d'autres sites tel que Mondovino (http://www.Mondovino.com/) et Vivino (https://www.vivino.com/FR/fr/) cependant :

- Vivino, tous les bots sont interdits dans le robots.txt.
- Mondovino : la gamme vin n'est pas complète, petite base de données, et souvent en doublon avec les vins que l'on retrouve chez Coop.

Prétraitement

À partir des fichiers de données recueillies du prétraitement des données devra être réalisé :

- Reformater les données
- Nettoyer

Toutes les recettes et tous les vins seront, ensuite, indexés dans une base de données.
Les Figure 1 et Figure 2 présente les schémas des objets Json contenant les données. Ceux-ci sont contenu dans des fichiers Json line (.jl).

Description des données

Données de Marmiton

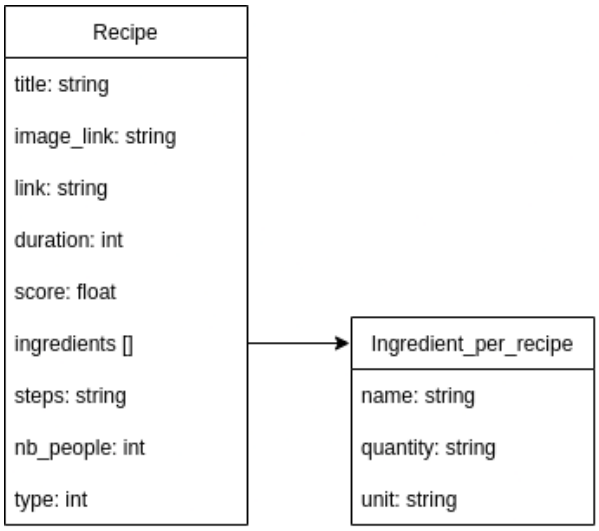


Figure 1: Schéma de données pour Marmiton

Les données de Marmiton sont réparties dans des objets contenant un array d'objet ingrédient. Si nous utilisons une base de données document, cela veut déjà dire qu'il y aura des relations one to many ou many to many.
Le champ steps contient les étapes que doit faire l'utilisateur pour réaliser la recette. Ce champ devra être traiter de façon plus précise car certains mots sont fusionnés et des erreurs durant le crawling rendent difficile la lisibilité.
Score correspond à la note des utilisateurs donné à la recette sur Marmiton. Nb_people correspond au nombre de personne que la recette peut nourrir. Type correspond au type comme dessert, plat, entrée.

Données de CoopVin

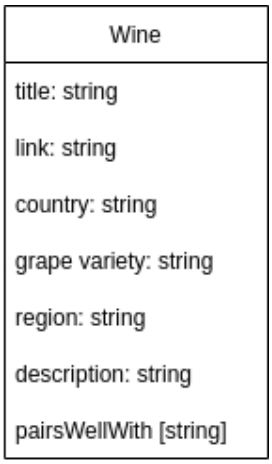


Figure 2: Schéma de données pour Coop vin

Pour Coop vin, les données utilisées sont semblables à Mondovino, Seules les plus importantes pour le projet sont conservées cependant. Si les 2 jeux de données sont conservés, nous utiliserons en priorité le format Coop car cela évitera les données manquantes. Ici aussi il y a une relation many to many avec pairWellWith (égal à Good_with).

Nous disposons de **32'149 recettes** (5'523 entrées, 16'907 plats, 19'719 desserts) et **1'652 vins** (451 pour les blancs, 1'237 pour les rouges).

Etat de l'art (projet similaire)

Il existe plusieurs projets sur les recettes de cuisine. Ils utilisent souvent leur propre base de données ou des bases de données benchmarks déjà connus.

Exploring Food Recipes using Word2Vec

Lien : https://www.kaggle.com/code/ajitrajput/exploring-food-recipes-using-word2vec/notebook
Dans ce projet kaggle, il nous est présenté comment utiliser word2vec pour trouver des similarités entres des ingrédients. Après avoir prétraité les différents ingrédients de plusieurs recettes, word2vec est entrainé dessus. Une fois ceci fait, les ingrédients sont comme placé dans un vecteur dans l'espace, donc un ingrédient proche d'un autre sera surement proche au niveau gout et utilité.

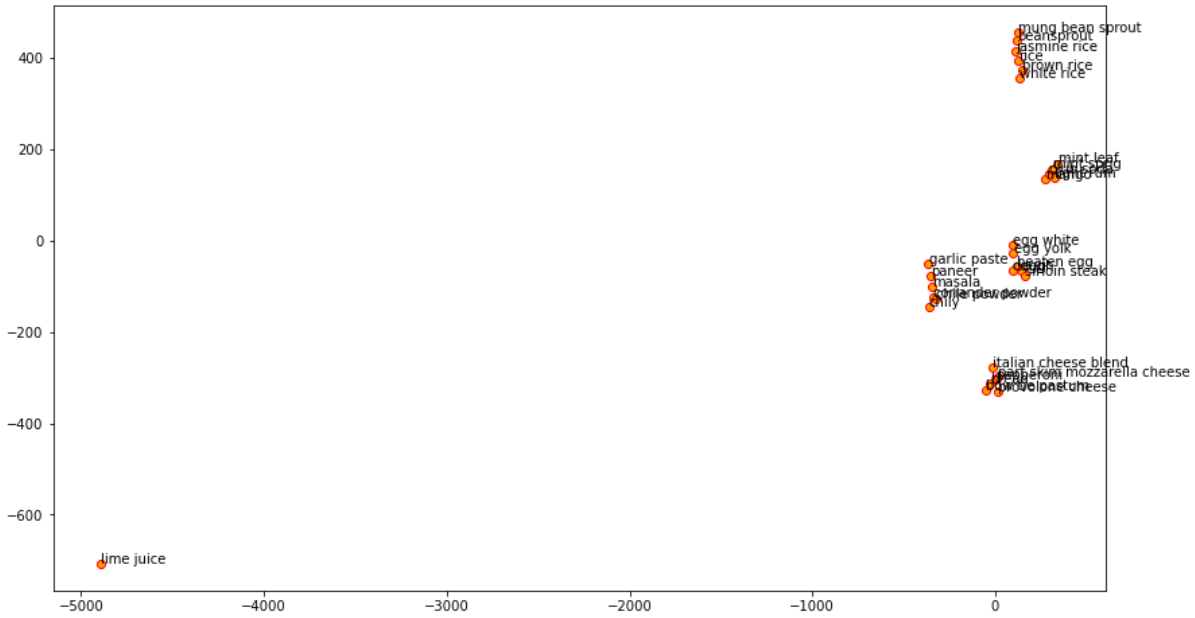


Figure 3: graphique montrant la séparation des recettes en clusters dans l'espace réduit en 2 dimentions

Par exemple sur la figure ci-dessus, on peut voir que le riz brun est proche du riz au jasmin.

How Dishes are Clustered together based on the Ingredients?

Lien : https://medium.com/web-mining-is688-spring-2021/how-dishes-are-clustered-together-based-on-the-ingredients-3b357ac02b26

Ici, les recettes sont regroupées en cluster à l'aide d'algorithme de réduction de la dimensionnalité. Par exemple ci-dessous, l'algorithme PCA réduit la dimensionnalité sur 2 dimensions.

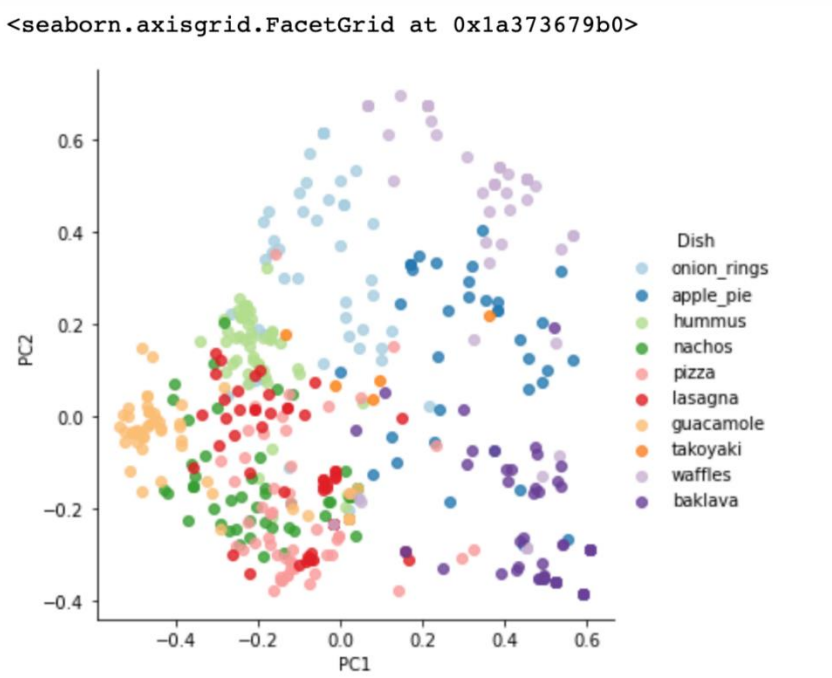


Figure 4: Graphique montrant la séparation des ingrédients dans l'espace

Sur la Figure 4 nous pouvons voir apparaitre des clusters distincts entre les familles de recettes.

Using machine learning to generate recipes that actually work

Lien: https://towardsdatascience.com/using-machine-learning-to-generate-recipes-that-actually-works-b2331c85ab72

Dans ce projet, les recettes sont utilisées pour nourrir différent générateur. Ceux-ci peuvent alors automatiquement retourner des recettes inédites.

Swissmilk

Lien : https://www.swissmilk.ch/fr/recettes-idees/

Le site Swissmilk propose une fonctionnalité similaire à celle du projet Eveille tes papilles. En donnant une liste d'ingrédients, des recettes sont proposées, voir figure ci-dessous.

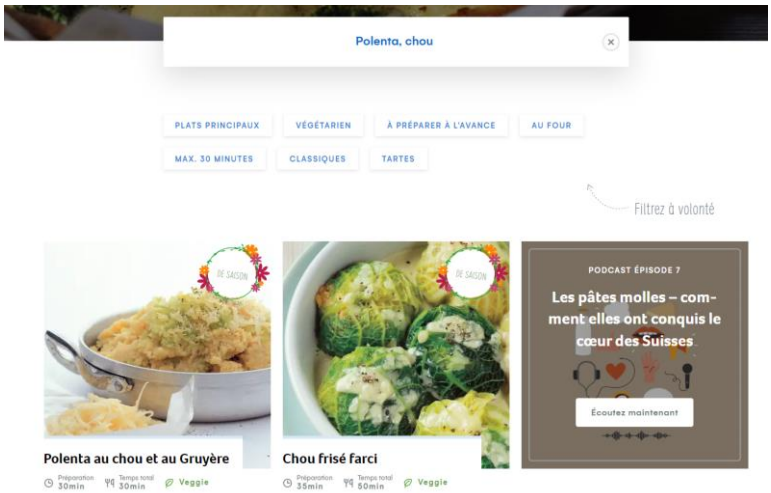


Figure 5 : Swissmilk idées recettes

Conception / Cas d'utilisation / Architecture

Création d'une application, tournant en local sans connexion internet, proposant à l'utilisateur des recettes de cuisines en fonction :

- De ses envies – filtrages : ingrédients, tant de cuissons, …
- Proposer des recettes en fonction des ingrédients à dispositions
- Proposer des recettes similaires aux recettes présélectionnées
- Proposer une gamme de vin par rapport aux recettes et vice-versa

L'utilisateur pourra en outre découvrir de nouvelles recettes dans sa gamme de goût, éviter le gaspillage alimentaire en utilisant les ingrédients à sa disposition et épater ses inviter avec des accords mets-vins incroyables.

Une interface graphique web sera créée pour donner une expérience conviviale à l'utilisateur.

Cas d'utilisation

- Un utilisateur peut trouver une recette qui correspond à certains critères, par exemple : Une personne veut chercher une recette pouvant être faite en 8 minutes avec la meilleure note possible.
- Un utilisateur qui a des ingrédients à sa disposition peut trouver une recette faisable avec ces ingrédients, par exemple :
    - Un utilisateur a des œufs, du fromage et du jambon, l'application devrait lui suggérer de faire une omelette au fromage avec du jambon.
- Un utilisateur faisant une recette peut trouver un vin qui irait bien avec.
- À partir de recettes que l'utilisateur apprécie, d'autres recettes peuvent lui être suggéré. Par exemple si l'utilisateur aime les lasagnes, le programme peut lui dire de faire des cannellonis à la sauce tomate.

Un utilisateur qui n'a pas un des ingrédients pour faire une recette, il peut demander à l'application de trouver un ingrédient proche pour remplacer. Par exemple : parmesan par du gruyère.

Architecture

Deux architectures distinctes sont envisageables pour la réalisation du projet. Toutes les deux permettent de réaliser une application locale simple mais elles offrent les 2 des avantages et des inconvénients déjà impactant à ce stade du projet.

Architecture RESTful

L'utilisation d'une API REST permet une plus grande évolutivité et scalabilité. Dans le cas où le projet doit évoluer sur d'autre plateforme, une API REST permettrait de répondre à cette demande.

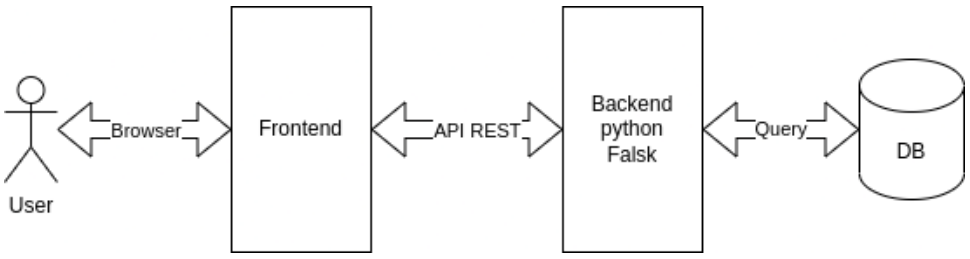


Figure 6: Diagramme d'architecture simple de type REST

API REST:
En fonction des fonctionnalités décrites dans Fonctionnalités, une ébauche d'API REST est présentée :

- Get/recipe param: score (note des utilisateur Marmiton), duration, nb people, …
- Get/recipe_by_ingredient param: ingredients: [ingredient, ingredient, ingredient, ...]
- Get/wine_pairing param: recipe: string
- Get/recipe_suggestion param: recipes: [recipe, recipe, ...]
- Get/ingredient_suggestion param: ingredients: [ingredient, ingredient, ingredient, ...]

Architecture UI

Une autre solution est de créer une application simple utilisant une librairie graphique, par exemple QT5. Cependant dans l'aspect évolutif du projet, cette solution est moins bonne.

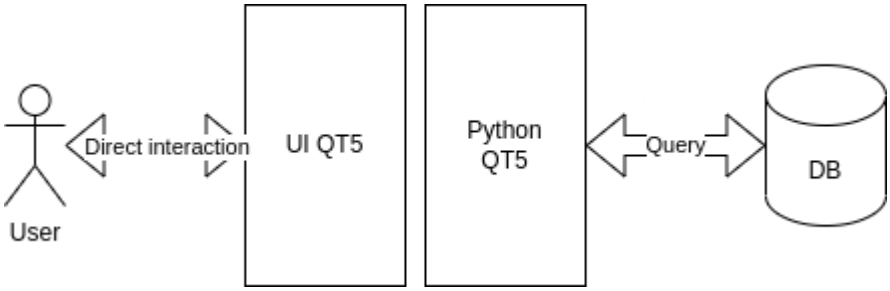


Figure 7: Diagramme d'architecture simple de type application QT

Une analyse plus poussée des architectures doit être réalisée pour un choix définitif.

Base de données

Dans les 2 solutions d'architectures, les données seront indexées dans une base de données. Là aussi deux choix de technologie sont possibles.

SQL (Relational)

Une base de données SQL permet de résoudre le problème des ingrédients par recettes (many to many) et aussi permet de formaliser la relation entre les vins et les recettes. Cependant il est difficile de faire du scoring avec les requêtes. La technologie envisagée pour cette architecture est **SQLite**. Il s'agit d'une base de données SQL qui ne consomme que très peu de ressource. Idéal pour les implémentations rapides en local ou les systèmes embarqué.

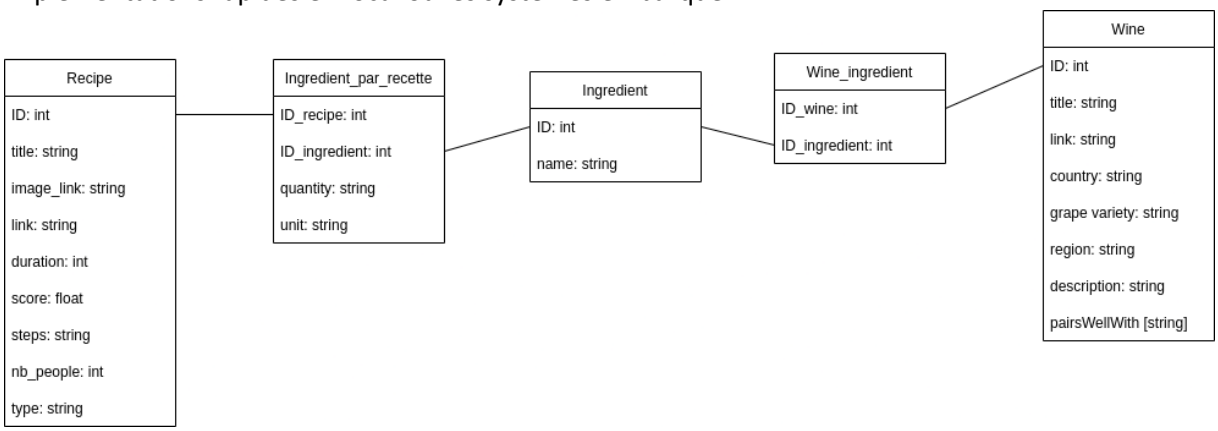


Figure 8: Schéma relationnel de la base de données SQL

NoSql (Document)

La solution NoSql permet de réaliser des requêtes avec du scoring cependant un problème majeur se pose c'est l'absence de relation many to many entre les ingrédients et les recettes, ce problème peut être en partie résolu si les ingrédients sont prétraités/lemmatisés (par exemple seulement œuf et non pas Oeuf, oeufs, petit œuf). **ElasticSearch** sera la technologie utilisée dans cette solution. Il s'agit d'une base de données de type Document, NoSql qui est très efficace pour les query avec scoring.

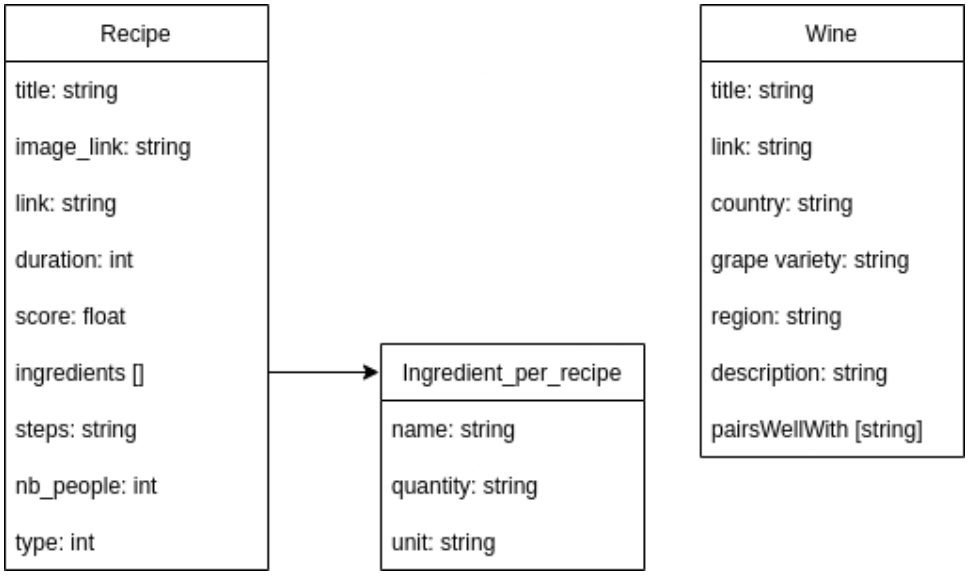


Figure 9: Schéma de la base de données de type Document

Exemple de query:
SELECT recipe WHERE ingrediens == User choice
SELECT recipe WHERE ingrediens == Vine.good_with

Architecture choisie

Dans cette partie est présenté les architectures potentielles. L'architecture Elastic Search a finalement été retenue. Voici maintenant une présentation de l'implémentation de l'application réalisée à partir de l'architecture conceptualisée.

Diagramme de classes final

En Figure 10, le diagramme de classes du backend. Il est composé de 3 classes Backend, DatabaseWrapper et Model. Le Backend agit en tant qu'API REST qui propose les 6 fonctionnalités différentes de l'application. La classe Database Wrapper comme son nom l'indique permet d'accéder à la base de données Elastic Search à l'aide de requêtes établies dans des méthodes de la classes. La classe Modèle est une classe python englobant le modèle word2vec.

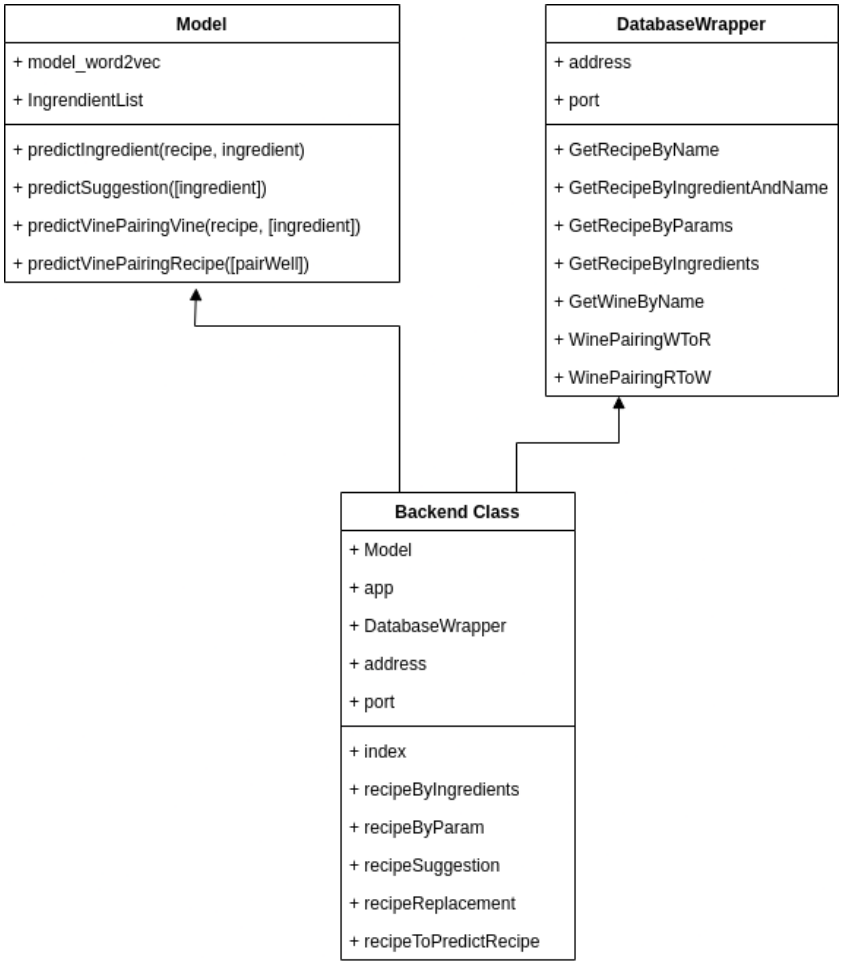


Figure 10 : Diagramme de classes du projet

Les différentes méthodes du DatabaseWrapper sont expliquées à la page 18 dans la section Query.

Diagramme architectural final

Ce diagramme présente le résultat final de l'application réalisée. Comme il est montré, les 4 composants principaux de notre application sont réunis dans un même réseau virtuel. Ils sont aussi tous « dockerizé » pour faciliter le déploiement. Le backend est accessible depuis la machine hôte sur le port 8080 grâce au binding de celui-ci. Le container du backend contient les modules du modèle et du wrapper de la base de données. Le frontend est lui accessible sur le port 3000 de la machine hôte. Il communique avec le backend directement sur le réseau virtuel.

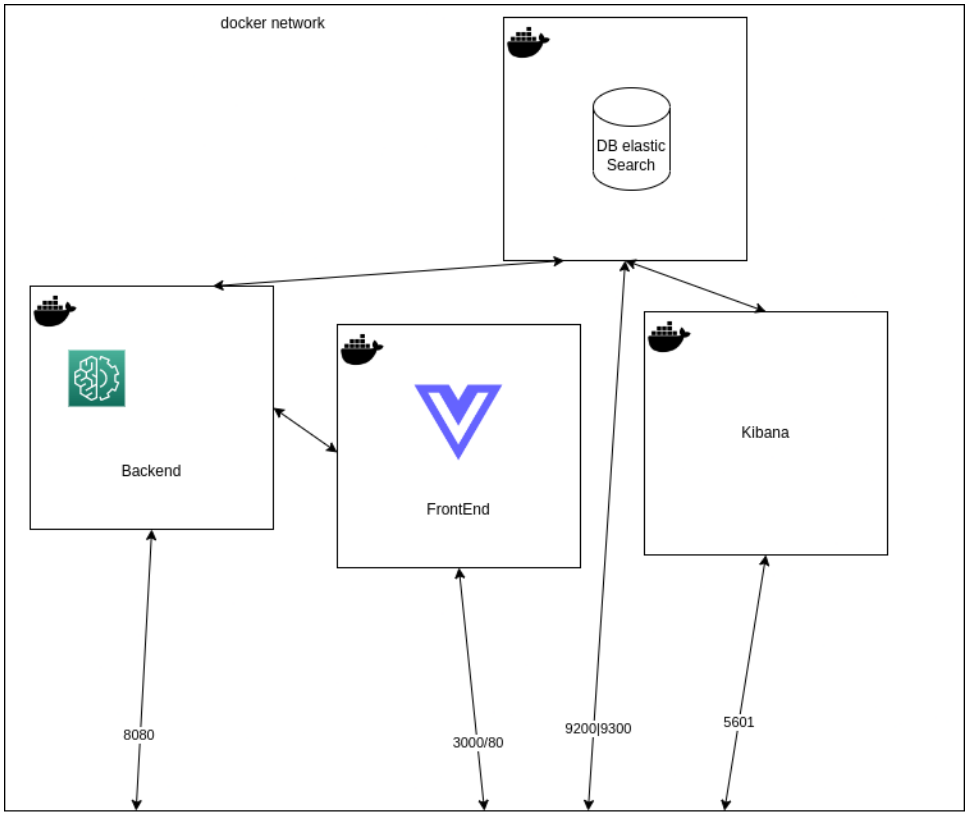


Figure 11 : Diagramme architectural

Cette architecture a été choisie pour permettre un déploiement rapide et simple et pour éviter les problèmes de compatibilité entre les machines hôtes sur lesquelles elle doit être déployée.

Arborescence du repository

L'arborescence du repository est faite pour simplifier la compréhension du travail réalisé. Il y a 4 dossiers principaux :

- Backend : Il contient les fichiers pythons contenant les classes présentées dans le diagramme de classes. Il contient aussi des tests effectués sur les classes pour s'assurer de leur bon fonctionnement. Le dossier Model contient tous les fichiers utilisés pour le modèle word2vec. Un notebook permet de comprendre les étapes qui ont été réalisées pour le modéliser.
- Database: il contient les fichiers de données contenant les recettes. Un docker-compose.yml permet de déployer l'environnement une fois que l'image du backend a été build. Les fichier pythons d'upload permettent à l'host d'uploader les données sur la base de données elastic search. (L'host doit avoir le package python elastic search 7.17.0 installé. Ce qui est une faiblesse pour le déploiement.)
- Scrapy: contient les fichiers dossier pour réaliser le crawling, L'arborescence est celle standard créer par un scrapy init.
- Le dossier GUI contient l'interface graphique.
- Le fichier deploy.sh permet de déployer l'application en local. Mais la base de données et toujours déployée dans des containers docker.
- Deploy_container.sh permet de déployer l'application « dockerizée » ainsi que la base de données.

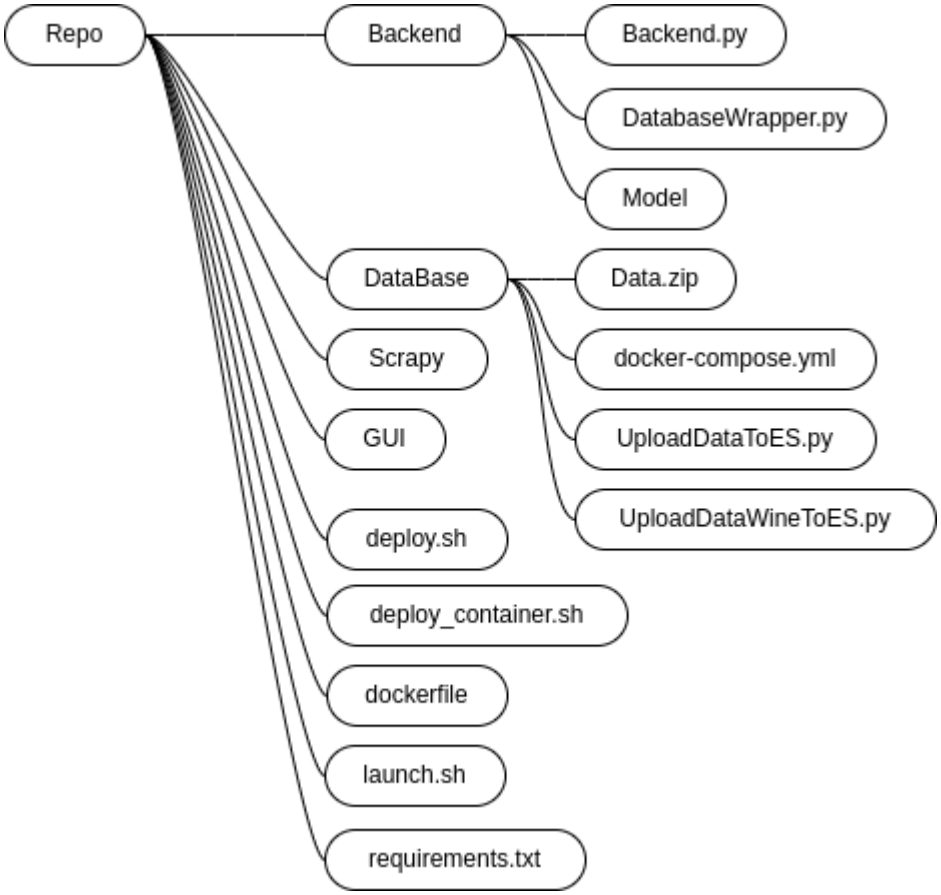


Figure 12 : arborescence du repository

Dépendances

Voici les librairies externes python dont dépend notre application. Elles sont présentées par classe :

- Model: numpy, difflib, matplotlib, sklearn, gensim
- DatabaseWrapper: elasticsearch
- Backend : falsk, waitress

Toutes ces dépendances sont inclues dans l'image docker mais pour uploader les données sur elastic search, il faut disposer du package elastic search 7.17.0 sur le host.

Fonctionnalités

Recherche de recettes :

- Trier les recettes en fonction de critères de recherches et filtrage

Proposition de recettes :

- En fonction des ingrédients à disposition de l'utilisateur, une liste de recettes est proposée. Possibilité également de filtrage

Wine pairing :

- Proposition de vins en fonction d'une recette. Cette proposition est obtenue grâce au descriptif des vins et les préférences en termes de cépages/régions de l'utilisateur

Suggestion de recettes :

- A partir de recettes « aimées » par l'utilisateur, l'application pourra suggérera des recettes pouvant lui convenir

Suggestion d'ingrédients :

- S'il manque un ingrédient pour une recette, l'application proposera un ingrédient de substitution ou rien si l'ingrédient n'a pas de substitue.

Techniques, algorithmes et outils utilisés

Le projet se décompose en 3 différentes parties :

1. Acquisition des données
2. Conception de l'outil
3. Test et validation

Certaines sont dépendantes d'autres mais 3 parties peuvent être faites en parallèle : UI, Database et Méthode d'analyse.

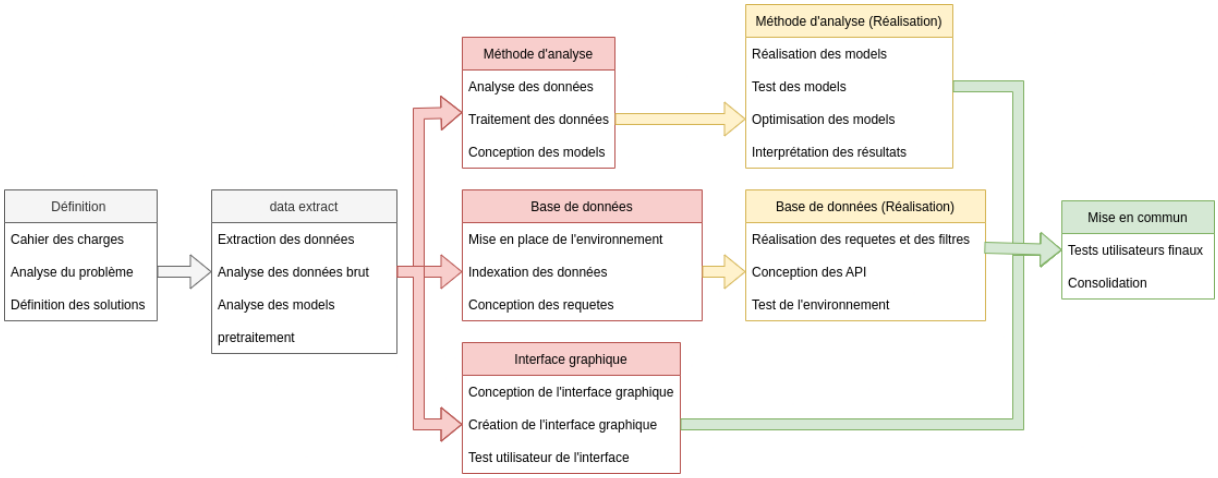


Figure 13: Pipeline du projet

Ci-dessous les parties 1 et 2 sont développés.

Acquisition des données

Voici les étapes qui ont été réalisées pour l'obtention des données depuis les sites précédemment décrit.

Crawler Marmiton

Le site Marmiton présente les recettes comme ci-dessous. Une page principale présente le nom de la recette le score, etc.… Le crawler va, pour chaque recette, appeler la fonction parse inner qui va, comme son nom l'indique, parser la page sur laquelle la recette pointe.
Les informations présentent sur la page principale sont transmises à la fonction parse inner dans ces arguments. Ces informations sont entre autres : le titre, le lien de l'image, le lien de la page de la recette, la durée de la recette et le score.
Dans la page de la recette, la fonction inner parse va elle, récupérer les ingrédients, leur quantité, les étapes pour la préparation de la recette le nombre de personnes.
Le champ type de recette est défini lors de l'appel à scrapy.
Une fois toutes les informations récupérées, la fonction inner parse écrit une ligne dans le fichier json line.
La fonction parse, une fois qu'elle a récupérée toutes les recettes de la page, appelle la fonction parse sur la page suivante.
Ceci est exécuté jusqu'à ce qu'il n'y ait plus de pages suivantes.

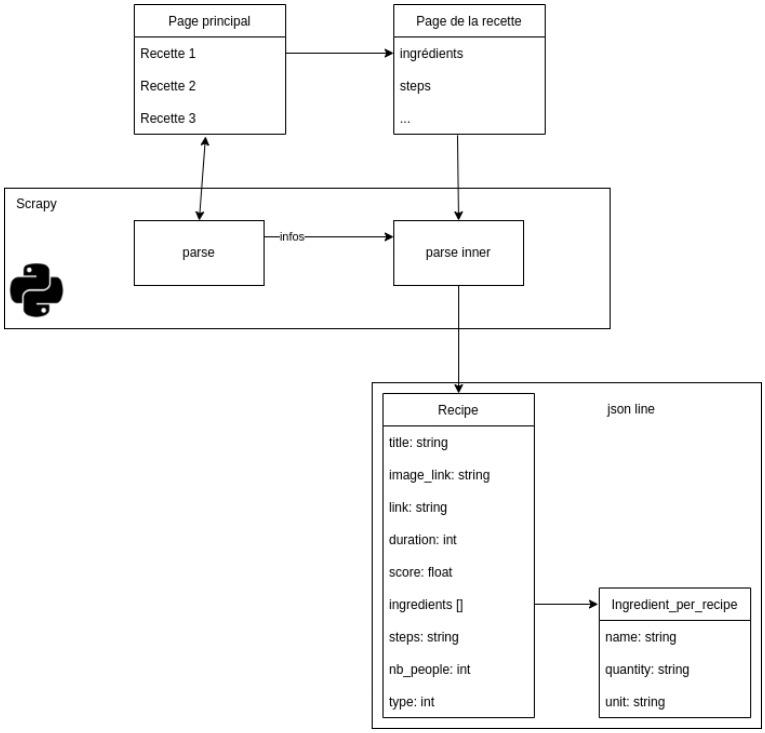


Figure 14 : Schéma du processus de crawling du site Marmiton

Crawler CoopVin

Là aussi comme pour Marmiton, la rubrique des vins de coop possède une page principale et une page pour chaque vin. Dans la même optique, la fonction parse parcourt les pages principales et appelle la fonction parse wine pour chaque vin. En lui passant le lien sur le chaque page en argument.

Les autres informations sont obtenues dans la page du vin. Une fois toutes les informations récupérées, une ligne est générée dans le fichier json line.

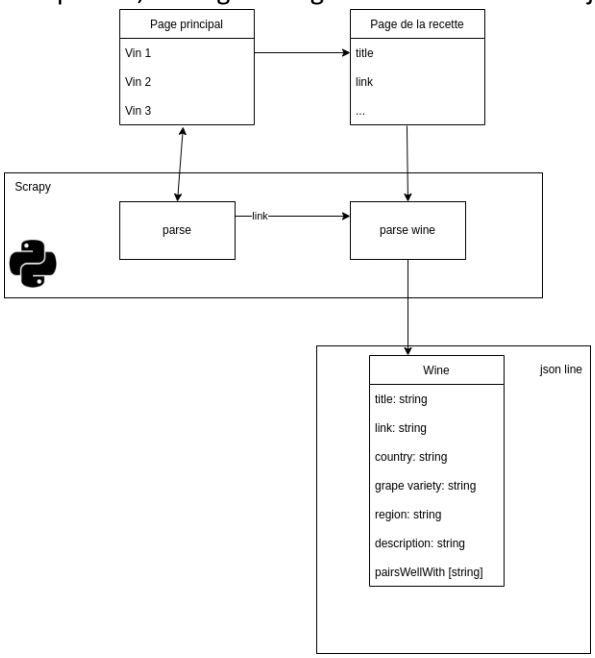


Figure 15: Schéma du processus de crawling du site coop vin

Database/Elasticsearch

Query

Voici les méthodes qu'implémente la classe databasewrapper. Chacune de ces méthodes execute une query Elastic Search sur la base de données.

GetRecipeByName

Cette query est utilisée par le backend pour les requêtes : /Recipes/Suggestion/, /Wines/Pairing/ et /Recipes/Pairing/. Elle permet de trouver la recette la plus similaire à celle que l'utilisateur entre pour éviter d'avoir une liste de recette à afficher. Par exemple si l'utilisateur veut une suggestion il peut simplement entrer le champ lasagne est le backend trouvera la recette la plus similaire. En utilisant la recette trouvée, des suggestions pourront donc être proposée.

Description de la requête :

Le nom donné par l'utilisateur doit **matcher** le champ **title** avec une **fuzziness** de 2 (acceptance des fautes de frappe)
Les réponses devraient (**should**) avoir un **score** (champ) de minimum 3 (ce filtre est **boosté**).

GetRecipeByIngredients

Cette query est directement utilisée par le backend pour trouver les recettes par ingrédients. Elle prend comme paramètres : un tableau contenant les ingrédients et un type de recette (déssert, plat, entrée). Un utilisateur peut donc trouver des recettes à partir des ingrédients qui la compose.

Description de la requête :

Pour chaque ingrédient, une requête **nested** est créé. Le nom de l'ingrédient doit **matcher** le champ **ingredients.ingredient** avec une **fuzziness** de 2.
Un **filtre** est appliqué. Seul (**must**) les recettes matchant le type sont retournées.
Les recettes sont triées par nombre d'ingrédient **ASC** suivi par le **_score** (qualité du hit) pour éviter que l'utilisateur n'ai une recette avec des ingrédient qu'il ne possède pas.

GetRecipeByIngredientAndName

Utilisée pour la requête /Recipes/Replacement/, elle permet de retourner une recette garantissant contenir un ingrédient. Ces paramètres sont simplement le nom de la recette et le nom de l'ingrédient qu'elle doit contenir.

Description de la requête :

Le nom de la recette doit **matcher** le champ **title** avec une **fuzziness** de 2.
Une query **nested** doit **matcher** le champ **ingredients.ingredient** avec le nom de l'ingrédient là aussi avec une **fuzziness** de 2.

GetRecipeByParams

Cette méthode appelé directement par le backend. Elle permet de rechercher des recettes en fonction de paramètres par exemple la durée de préparation, le type, le score, …

Description de la requête :

Pour chaque paramètre passé (la clé) 2 valeurs lui sont associé (la valeur minimale et maximale)
Pour chaque clé une query est créer où le paramètre lié à la clé doit être plus grand que la valeur minimal (**gte**) et moins grand que la valeur maximale (**lte**).

GetWineByName
Cette méthode est appelée par le backend pour la requête : WineToPredictRecipe. Comme la méthode getRecipeByName, celle-ci permet de trouver le vin avec le nom le plus proche entré par l'utilisateur.

Description de la requête :
Le nom donné par l'utilisateur doit **matcher** le champ **title** avec une **fuzziness** de 2.

WinePairingWToR
Cette méthode permet en fonction des pairwellswith d'un vin, de trouver une recette qui possède ces caractéristiques. Cette fonctionnalité est provisoire pour tester le système. Elle est normalement implémentée par le modèle word2vec.

Description de la requête :
Pour chaque pairwells, les termes qui le compose sont splités. Par exemple : "saucisse au chou" devient "saucisse" "au" "chou". Chaque terme devrait ensuite matcher le plus que possible les ingrédients d'une recette.

WinePairingRToW
Cette méthode permet d'obtenir un vin à partir des ingrédients d'une recette. Là aussi cette méthode est provisoire.

Description de la requête :
Chaque ingrédient devrait **matcher** un terme dans un **pairwellswith**.**term**

Schéma

Voici, en Figure 16, le schéma des données dans Elastic Search. Il ressemble fortement à celui conceptualisé mais il contient quelques différences. Par exemple les types. En effet, Elastic Search utilise le type « text » pour des textes longs comme les steps par exemples. Le type keyword est parfait pour des champs contenant un ou quelques mots. Elastic Search optimise certaines requêtes sur ce type de champs.
Les ingrédients sont des types nested c-à-d. qu'ils contiennent des champs enfants. Aussi sans ce type, il est impossible de faire des queries sur les enfants.
Un champ a été ajouté au schéma des recettes. Il s'agit du champs ingredient_count qui permet de connaitre le nombre d'ingrédients par recettes. Sans ce champ, il faut créer une query script contant le nombre de champ nested ce qui est contraignant et couteux en temps de recherche.
Pour les vins les pairsWellWith sont aussi devenu des champs de type nested. Pour la même raison que les ingredients.

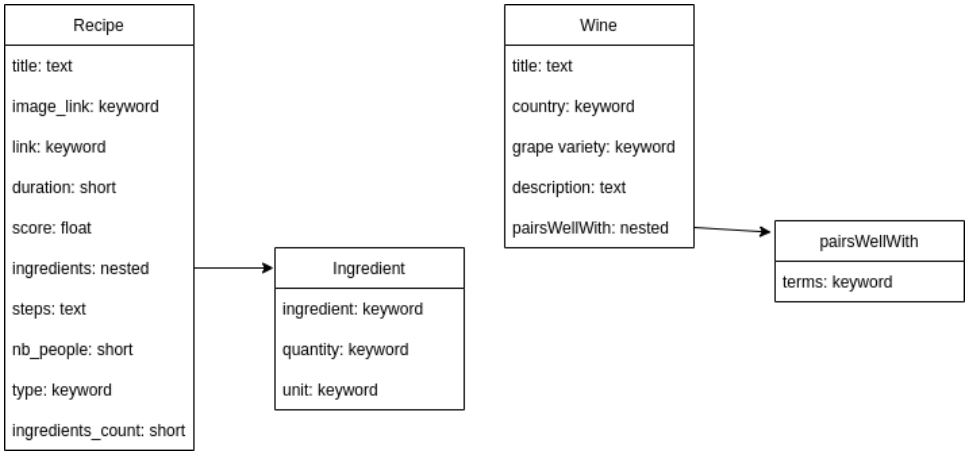


Figure 16 : Schémas de données d'Elastic Search

Backend

Le backend expose une API REST pour le frontend. Il dispose des 6 fonctionnalités de base établies précédemment. À savoir : obtenir une recette depuis des ingrédients, obtenir une recette depuis des critères, obtenir une suggestion d'ingrédient qui irait potentiellement bien avec la recette, obtenir une suggestion d'ingrédient pouvant remplacer l'ingrédient indiqué, trouver une recette allant bien avec un vin et trouver un vin allant bien avec une recette.

API REST

Voici la description de cette API

GET /
Il s'agit de l'index principal. Il ne retourne aucune fonctionnalité juste un message pour montrer que le serveur est vivant.

Requête exemple :
http://localhost:8080/
Retourne : {'Something is happening!?!?!': 'I AM ALIVE!!!!'}

GET /Recipes/ByIngredients/
Cette requête retourne des recettes utilisant les ingrédients entrés par l'utilisateur. La recette avec le moins d'ingrédient est la première. Si aucune recette n'existe avec les ingrédients entrés, le système retourne celle avec le moins d'ingrédients en plus à ajouter.
Elle n'utilise que la méthode GetRecipeByIngredients du databaseWrapper

Requête exemple :
http://localhost:8080/Recipes/ByIngredients/?type=dessert&ingredients=whisky,farine
Retourne : recipes [les 3 recettes avec le meilleur hit]

GET /Recipes/ByParam/
Cette requête permet d'obtenir les meilleures recettes selon des critères établis par l'utilisateur. Là aussi elle n'utilise que la méthode GetRecipeByParams du databaseWrapper

Requête exemple :

http://localhost:8080/Recipes/ByParam/?type=dessert&duration=30,120&nb_people=2,6&score=2,4
Retourne : recipes [les 3 recettes avec le meilleur hit]

GET /Recipes/Suggestion/
Ici elle fournit une suggestion d'ingrédient qui irait potentiellement bien avec le plat. Comment l'incorporé n'est pas établi cependant. Elle fait appelée à la méthode GetRecipeByName du databaseWrapper pour trouver la recette avec le nom le plus proche que celle entré par l'utilisateur puis la méthode predictSuggestion du modèle pour trouver une correspondance.

Requête exemple :
http://localhost:8080/Recipes/Suggestion/?recipe=tarte_au_pomme
Retourne : {'ingredients': les meilleures prédictions d'ingrédient}

GET /Recipes/Replacement/
Elle permet de remplacer un ingrédient qui n'est pas à la disposition de l'utilisateur par un proche. Exemple : paprika par poivre de cayenne. Elle utilise aussi des méthodes du databaseWrapper (GetRecipeByIngredientAndName) et du modèle (predictIngredient).

Requête exemple :
http://localhost:8080/Recipes/Replacement/?recipe=flan_au_chocolat&ingredient=chocolat
Retourne : {'ingredients': les meilleures prédictions d'ingrédient}

GET /Wines/Pairing/
Pour le wine pairing l'utilisation du modèle n'a pas pu être réalisée à temps donc le système utilise seulement des queries sur la base de données. Là le backend utilise les méthodes GetRecipeByName et WinePairingRToW du databaseWrapper.

Requête exemple :
http://localhost:8080/Wines/Pairing/?recipe=flan_au_chocolat
Retourne : {'wine': les 3 meilleurs vins allant avec la recette}

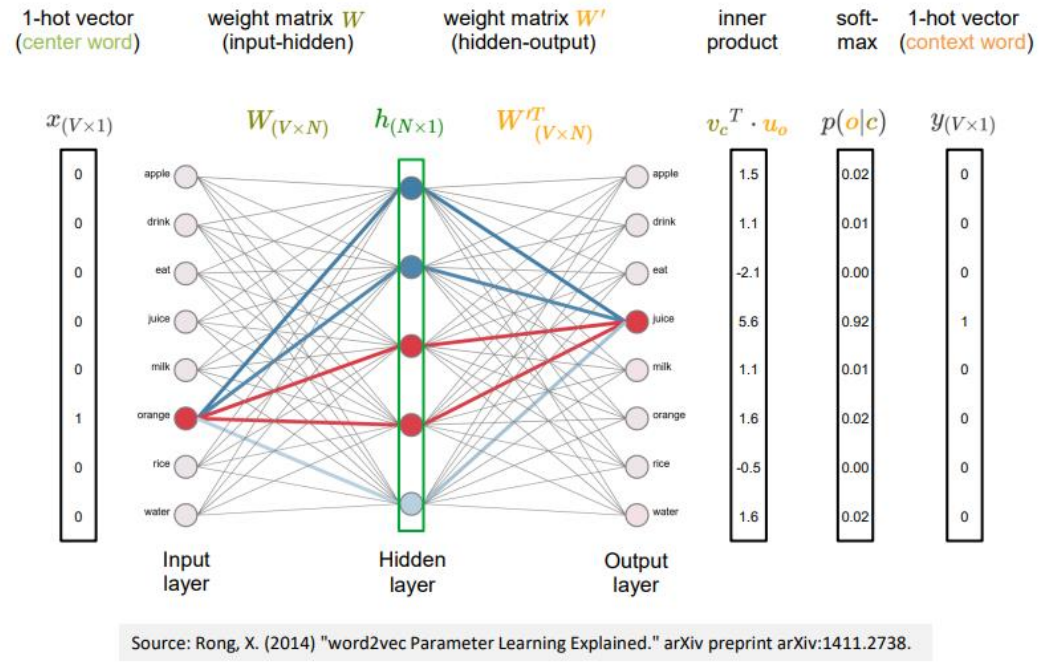
GET /Recipes/Pairing/
Là aussi le modèle utilise seulement des méthodes du databaseWrapper: GetWineByName et WinePairingWToR.

Requête exemple :
http://localhost:8080/Recipes/Pairing/?wine=lavaux
Retourne : {'recipe': les 3 meilleures recettes allant avec le vin}

Suggestion d'aliments

Modèle Word2Vec

Word2Vec est un modèle utilisé pour le plongement lexical – word embeddings. Le principe du modèle est, comme son nom l'indique, de transformer des mots en vecteurs. Nous utiliserons ces vecteurs pour placer les ingrédients dans un espace à N dimensions et ainsi pouvoir trouver des similitudes entre eux. Ceci nous permettra de proposer à l'utilisateur un aliment en fonction d'une recette ou d'une liste d'aliments.



Source: Rong, X. (2014) "word2vec Parameter Learning Explained." arXiv preprint arXiv:1411.2738.

Figure 17 : Visualisation du réseau Word2Vec

Dans sa forme la plus simple, le but du modèle est de prédire un mot en sortie en fonction du mot en entrée. Cependant ce n'est pas cette fonction qui est intéressante, car la prédiction n'est pas très bonne, mais ce sont les embeddings – les vecteurs composants les matrices W et W' – qui vont nous intéresser.

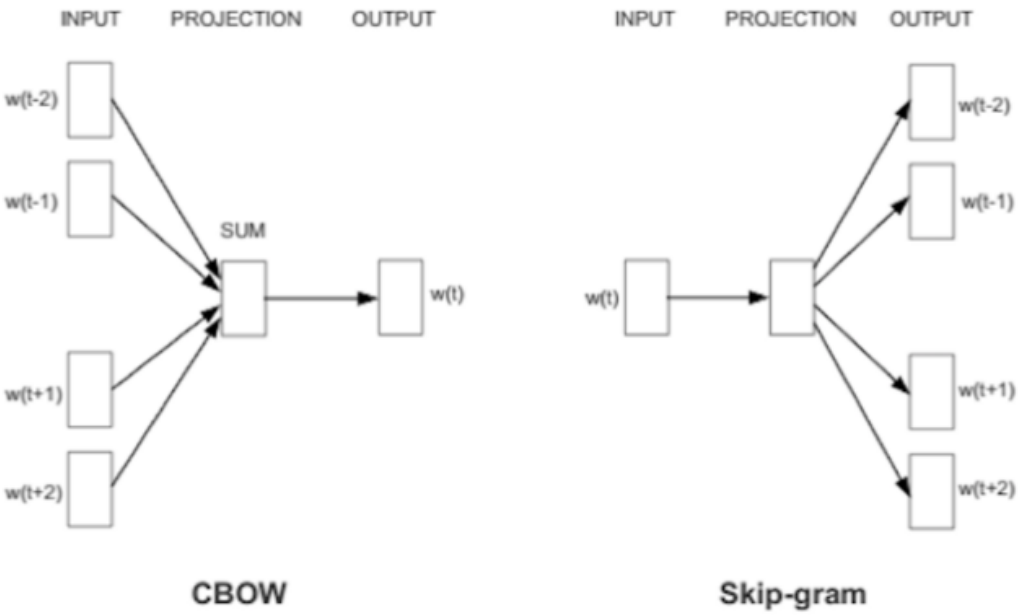
Les entrées et sorties sont en « on-hot-encoding » c'est-à-dire qu'il y a une entrée et une sortie pour chaque mot du vocabulaire.

Après entrainement du modèle, les mots en entrée vont être représentés par leur embedding. De cette manière, il est possible de positionner les mots dans une espace à N dimension, où N est défini par le nombre de neurones dans la couche cachée. A l'état de l'art N vaut 300, mais une couche cachée de 100 neurones est également fréquemment utilisée.

Pour augmenter l'efficacité du modèle, il existe deux autres variantes : CBOW et SGNS. Au lieu d'avoir un unique mot en entrée et de même pour la sortie, ces deux méthodes s'intéressent également aux mots à proximité.

1. CBOW : Méthode par sac de mots. Cette méthode s'appuie sur un mot en entrée plus les mots l'entourant pour prédire la sortie

2. SGNS : Méthode Skip-Gram avec un « negative sampling ». Cette méthode n'a qu'un mot en entrée et va donner les probabilités sur plusieurs mots en sortie. Le mot souhaité et ceux qui l'entoure. De plus il donnera de points négatifs à certains mots (aléatoirement) ne l'entourant pas.



Images by Tomas Mikolov et al. 2013

Figure 18 : Word2Vec - CBOW et Skip-Gram

Dans le cadre de notre projet nous allons nous intéresser à la méthode CBOW. Bien que la méthode SGNS soit très efficace, Word2Vec utilise par défaut CBOW. Si cette méthode n'avait pas donné de résultat concluant, des essais auraient été mené sur SNGS.

Etant donné que l'entrée du modèle est en « one-hot » et que les données à dispositions, les recettes, sont en format « texte », il faut effectuer un prétraitement pour tokeniser et lemmatiser les recettes. Ces deux étapes seront réalisées à l'aide de l'API Spacy qui permet de faire de la lemmatisation en français grâce à son modèle « fr_core_news_sm ». Une dernière étape, avant de pouvoir entrainer le modèle, sera d'enlever les « stop words » c'est-à-dire les mots n'apportant pas de sens tel que : le, la, du, des, l', et, dans, …

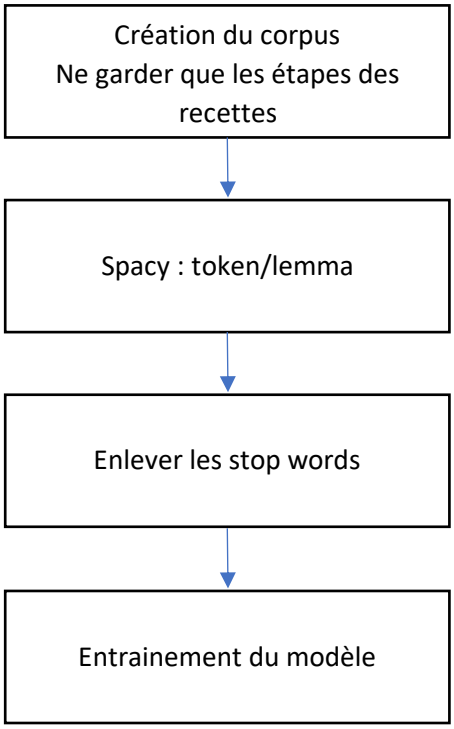


Figure 19 : Word2Vec - Etape prétraitement

Le modèle Word2Vec est issu de l'API Gensim. L'entrainement se fait de la façon suivante :

```
model = Word2Vec(sentences=corpus, vector_size=100, window=5, min_count=1,
workers=4)
```

Où:

- Sentences est notre corpus, donc nos recettes sous forme de token
- Vector_size est la taille de la couche cachée, ce qui représente également la dimension des embeddings – vecteurs
- Window est le nombre de mots pris en compte. C'est-à-dire pour le modèle CBOW le mot dont on souhaite prédire la suite + 2 mots avant et 2 mots après
- Min_count est le nombre de fois au minimum qu'un mot doit apparaître pour être tenu en compte
- Workers est le nombre de partition durant l'entrainement
- Sg est le modèle : CBOW ou SGNS utilisé. Par défaut le modèle est CBOW et donc le champ n'est pas renseigné.

Pour gagner en place mémoire, une fois le modèle entrainé, nous ne sauvegarderons en mémoire que les vecteurs qui seront chargés au démarrage de l'application.

Fonctions de suggestions

L'avantage de Word2Vec et de la méthode CBOW est qu'un mot est prédit selon plusieurs mots, donc dans un contexte, et qu'il est ensuite placé dans l'espace, selon notre entrainement, à 100 dimensions. L'utilisation du contexte donnera une forte valeur ajoutée au résultat final car des ingrédients seront proches dans l'espace non seulement pour leur caractéristique « biologique » par exemple : légume vert, mais également par la façon de cuisiner : couper, râper, …

Suggestion d'un ingrédient selon une liste

La suggestion d'un ingrédient se fait se fait simplement sur la base des ingrédients reçus en argument. Cette méthode calcule la similarité, en cosinus, entre une simple moyenne des vecteurs des clés données et les vecteurs pour chacune des clés du modèle.
Méthode Word2Vec : most_similar(positive=None, negative=None, topn=10, clip_start=0, clip_end=None, restrict_vocab=None, indexer=None)

La méthode retournera une liste de 5 ingrédients qui se marieront bien avec les ingrédients en entrée.
Comme les ingrédients sont reçue depuis la base de données il faut s'assurer que les mots de la liste reçue soient bien dans les clés du modèle. Un test est donc réalisé avant d'appeler la méthode de similarité.

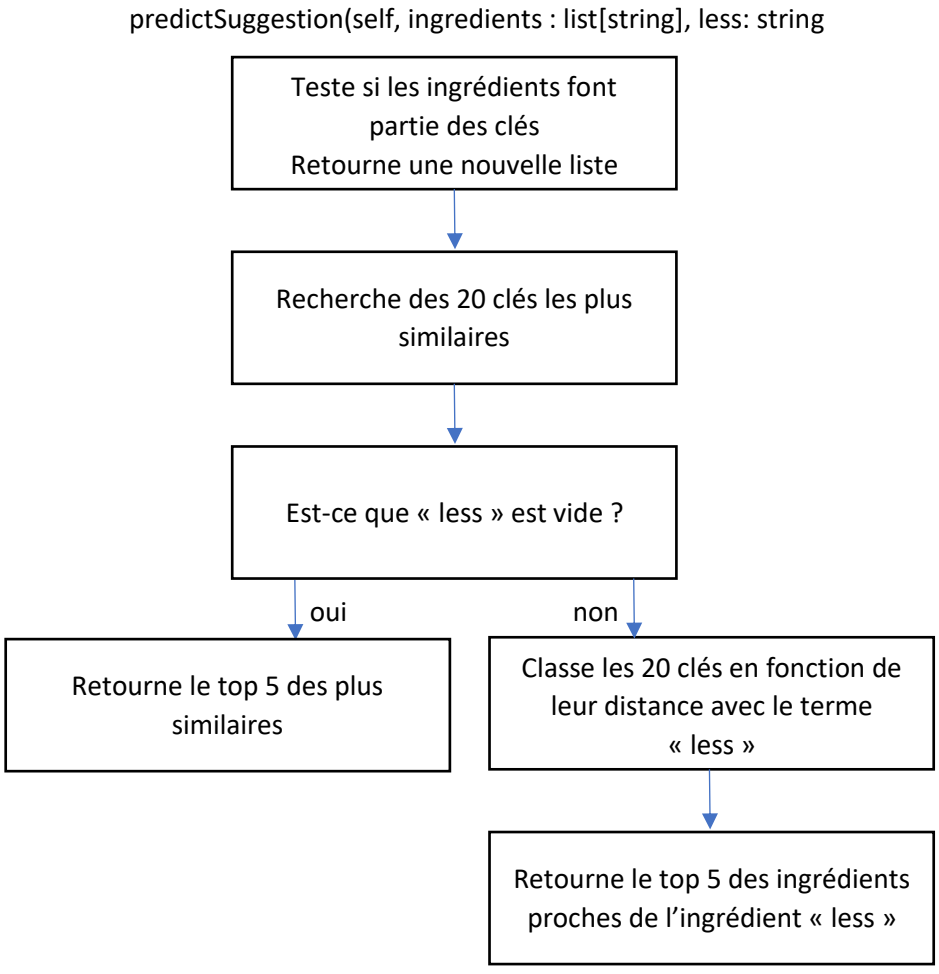


Figure 20 : Méthode predictSuggestion

Suggestion d'un ingrédient de remplacement

Pour la suggestion de remplacement d'un aliment, ce n'est pas une liste d'ingrédients qui sera passée comme argument mais les étapes de la recette. De cette manière nous retournerons une liste d'ingrédients allant bien dans la recette. De plus, l'utilisateur pourra indiquer un ingrédient qu'il souhaite remplacer dans sa recette.

Cette méthode utilisera la méthode predictSuggestion pour retourner la liste des ingrédients en fonction de leur similarité et de la distance avec l'ingrédient à remplacer.

Une étape supplémentaire doit être faite par rapport à la « simple » suggestion, c'est celle de la tokenisation et lemmatisation de la recette pour en extraire les termes intéressants pour la suggestion.

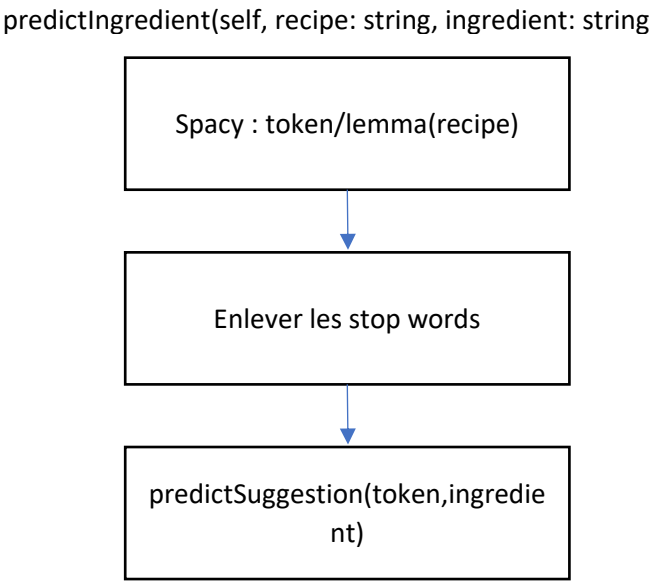


Figure 21: Méthode predictIngredient

Pour trouver l'ingrédient de remplacement, la méthode **distances** de Word2Vec est utilisée. Cette méthode calcule la distance cosinus entre deux clés. Ici nous donnons en entrée une liste de clé à comparer avec une clé, l'ingrédient que l'on désire changer. La méthode retournera une liste de distance qu'il faut ensuite trier. Tout ceci est fait dans la méthode predictSuggestion lorsque l'on donne une clé « less ».

Interface Web

L'interface web propose plusieurs pages permettant de facilement utiliser les différentes fonctionnalités présentes dans le backend. Chaque page propose un formulaire permettant de saisir les différents paramètres nécessaires pour exécuter la requête vers le backend. Ensuite, les résultats sont présentés à l'utilisateur.

Recherche par ingrédients

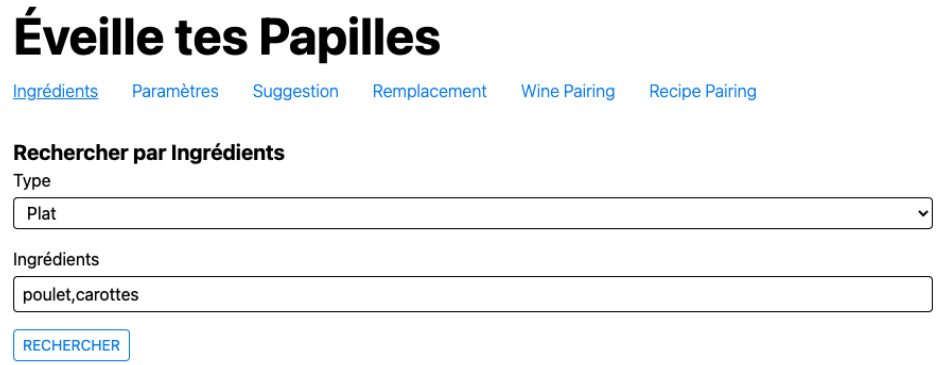


Figure 22 : GUI de recherche par ingrédients

L'utilisateur peut choisir le type de repas (entrée, plat ou dessert) et les ingrédients qu'il souhaite utiliser (sous forme de liste séparée par des virgules).

Recherche par paramètres

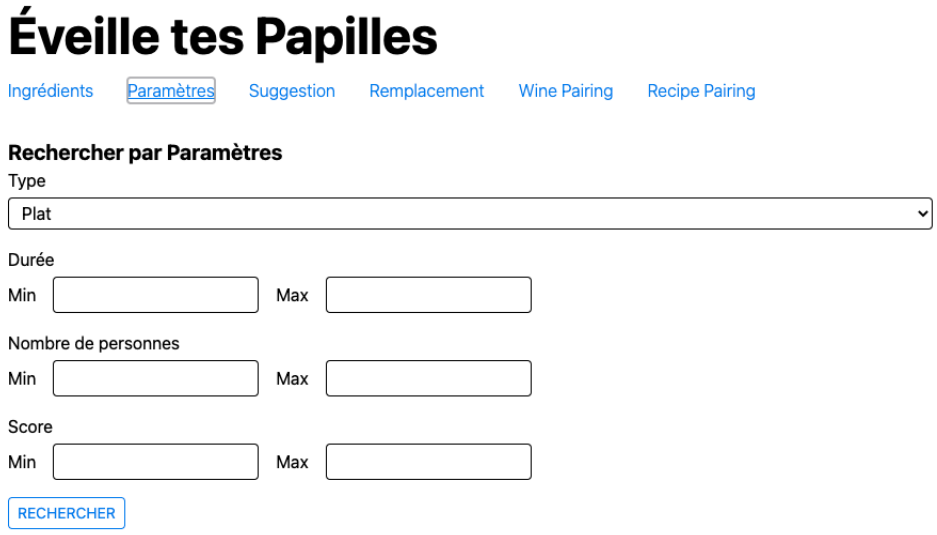


Figure 23 : GUI de recherche par paramètres

L'utilisateur peut rechercher des recettes grâce divers paramètres. Il peut saisir les valeurs minimums et maximum pour chacun des paramètres.

Suggestion d'ingrédients

Figure 24 : GUI de suggestion d'ingrédients

L'utilisateur reçois une suggestion d'ingrédients qui irai bien avec la recette saisie.

Remplacement d'ingrédients

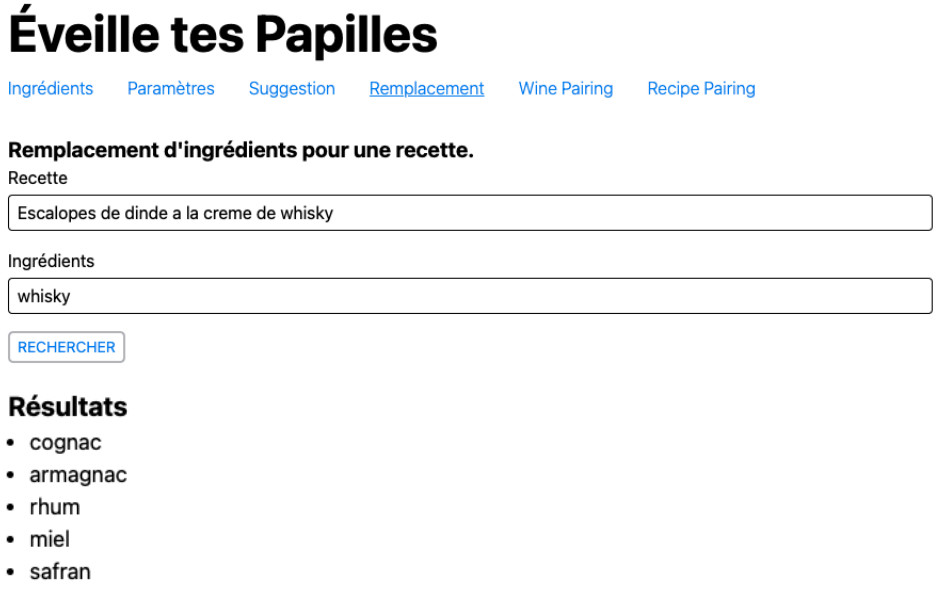


Figure 25 : GUI de remplacement d'ingrédients

L'utilisateur reçois une suggestion d'ingrédients qui irai bien avec la recette saisie.

Wine Pairing

Figure 26 : GUI de Wine Pairing

L'utilisateur peut rechercher des vins qui s'accordent bien avec la recette saisie.

Recipe Pairing

Figure 27 : GUI de Recipe Pairing

L'utilisateur peut rechercher des recettes qui s'accordent bien avec le vin saisi.

Représentations des recettes et des vins.

Figure 28 : GUI de recettes

Les recettes sont représentées sous cette forme. Nous y retrouvons :

- Le nom de la recette
- Une photo
- Les ingrédients
- Les étapes de la préparation
- Le score
- Un lien externe contenant plus d'information (le lien peut être obsolète dans certains cas).

Figure 29 : GUI de vin

Les vins sont représentés sous cette forme. Nous y retrouvons :

- Le nom du vin
- La variété de grappe
- Le pays
- Une description
- Une liste d'ingrédients s'accordant bien avec le vin
- Un lien externe contenant plus d'information (le lien peut être obsolète dans certains cas).

Planification

Le gantt rempli est en annexe de ce document. Nous n'avons pas eu trop de mal à respecter les contraintes de temps et les tâches du gantt. La vraie difficulté a été d'alterner avec les autres projets du cursus. Nous n'avons aussi pas eu le temps de réaliser beaucoup d'optimisation de notre système.

Test et validation

Modèle Word2Vec

Un notebook jupyter de test a été réalisé. Ce notebook permet de tester les différentes méthodes implémentées en plus du modèle Word2Vec réalisé.

Essai du modèle :

```
model.wv.most_similar(['poulpe'])
```

```
[('tentacule', 0.6727848649024963),
 ('calamar', 0.6492988467216492),
 ('encornet', 0.6381126046180725),
 ('calmar', 0.6257791519165039),
 ('tripe', 0.6118736267089844),
 ('rognon', 0.6025072336196899),
 ('seiche', 0.5921154022216797),
 ('langouste', 0.5822075009346008),
 ('cervelle', 0.5755167007446289),
 ('betterave', 0.5706781148910522)]
```

```
model.wv.most_similar(['roquette'])
```

```
[('mache', 0.8494817018508911),
 ('mesclun', 0.803922176361084),
 ('salade', 0.7655790448188782),
 ('laitue', 0.7473024129867554),
 ('menthe', 0.7182823419570923),
 ('cerfeuil', 0.7102077007293701),
 ('aneth', 0.6737106442451477),
 ('pluche', 0.6579697132110596),
 ('decore', 0.6451163291931152),
 ('coriandr', 0.6432075500488281)]
```

Figure 30 : Test des similarités

```
model.displayScatterplot(model,['poulet','carotte', 'oignon',
                                'soja', 'ail', 'escalope',
                               'bouillon', 'eau', 'gingembre',
                               'citronnelle', 'coriandre', 'riz', 'gingembre', 'citronnelle'])
```

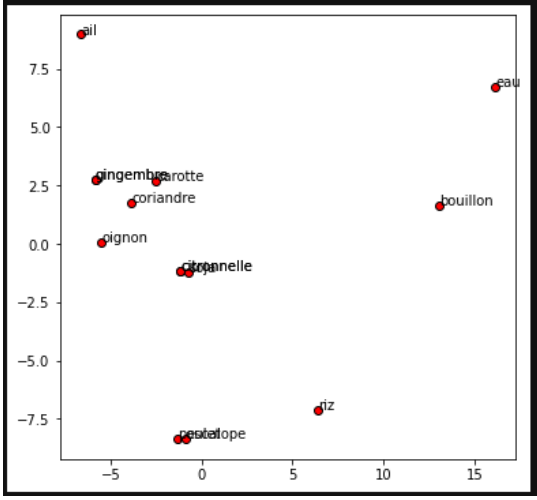


Figure 31 : Visualisation de certaines clés

Sur la figure ci-dessus, en projetant différentes clés/embeddings dans un espace 2D en effectuant une PCA, on voit le lien entre les ingrédients. Ainsi on voit que l'eau et le bouillon sont relativement proches mais éloignés des autres, ce qui fait sens. De plus poulet et escalope sont quasiment confondu et les ingrédients servant à donner du goût tel que la citronnelle et le soja mais aussi le gingembre, entre autres, sont très proche. L'ail se retrouve un peu éloigné, bien qu'il donne également du goût, mais la projection 2D à partir de vecteurs à 100 dimensions n'est pas toujours très représentatives : exemple si l'on n'affiche pas tous les ingrédients.

Essai de la suggestion d'un ingrédient en fonction d'une liste :

```
model.predictSuggestion(['spaghetti','lapin','pois','oranges', 'WEM'])
```

```
ingredients for the suggestion :  ['spaghetti', 'lapin', 'pois', 'orange']
['rhubarbe', 'ananas', 'poulpe', 'encornet', 'potiron']
```

Figure 32 : Test predictSuggestion

Le test effectué ci-dessus démontre les fonctionnalités de la méthode predictSuggestion, voir page 25. Lors du test il est possible de voir les ingrédients pris en compte pour la suggestion d'un nouvel

ingrédient. Si un ingrédient est mal orthographié ou au pluriel la fonction modifie l'orthographe en conséquence. Cependant si le mot ne figure pas dans le dictionnaire, dans les embeddings, ce dernier n'est pas pris en compte.

Essai du remplacement d'un ingrédient :

```
number = 785
recipe = model.recipeDta[1][number]['steps']
recipename = model.recipeDta[1][number]['title']
recipeningredients = model.recipeDta[1][number]['ingredients']
ingredient = model.predictIngredient(recipe,'poulet')
print("\n",recipename)
print(recipe)
print(ingredient)
```

```
ingredients for the suggestion :  ['carotte', 'oignon', 'poireau', 'pois', 'soja', 'ail', 'escalope', 'poulet', 'eau
', 'sel', 'bouillon', 'volaille', 'eau', 'gingembre', 'citronnelle', 'coriandre', 'poulet', 'riz', 'gingembre', 'citr
onnelle']

 Bouillon thai au poulet
Preparer les legumes, les eplucher et les couper en petits des Faire revenir les carottes, les oignons, les poireaux,
les pois plat, les pousses de soja et l'ail (bien hache) Saler, poivrer Ils doivent etre croquants Faire cuire les es
calopes de poulet et les couper en petits des Remplir un grand faitout d'eau, mettre du gros sel et les 2 cubes de bo
uillon de volaille Quand l'eau bout, y verser la sauce soja, la sauce nuoc-mam, le gingembre rape, la citronnelle (se
lon les gouts), la coriandre ciselee, saler et poivrer Ajouter les legumes et le poulet Pour finir mettre le vermicel
le de riz (bien l'emietter) Evidemment les dosages reste a votre appreciation, notamment pour le gingembre, la citron
nelle et les sauces
['boeuf', 'tofu', 'poivron', 'potiron', 'mangue']
```

Figure 33 : Test predictIngredient

La méthode predictIngredient parcourt la recette reçue et en défini une liste d'ingrédient. Ensuite elle retourne une liste d'ingrédient pouvant remplacer celui donné en entrée, ici « poulet ».

Database - Backend

Pour la partie database et backend des tests d'intégration ont été réalisé pour tester le bon fonctionnement du système avec le container Elastic search. Ils sont disponibles dans le dossier Backend.

Voici des résultats en fonction des différentes fonctionnalités implémentées par la base de données (les fonctionnalités du modèle word2vec sont déjà testées) :

Recipes/ByIngredients/

curl http://localhost:8080/Recipes/ByIngredients/?type=dessert&ingredients=whisky,farine

La requête ci-dessus cherche des recettes contenant les ingrédients whisky et farine

L'application nous retourne les recettes suivantes :

- Flan de courgettes en dessert
- Ananas rôti et son crumble au spéculoos

Les 2 recettes contiennent effectivement les ingrédient whisky et farine. Le flan contient moins d'ingrédients que la deuxième recette ce qui était attendu.

Recipes/ByParam/

curl http://localhost:8080/Recipes/ByParam/?type=dessert&duration=30,120&nb_people=2,6&score=2,4

La requête ci-dessus retourne des recettes d'une durée de réalisation de 30 minutes à 2 heures, pour 2 à 6 personnes et avec un score compris entre 2 et 4 étoiles. La recette doit aussi être un dessert.

L'application nous retourne les recettes suivantes :

- Macarons inratables : durée de préparation : 50 minutes, nombre de personnes : 4 et score : 3.5
- Paris-Brest généreux : durée de préparation : 75 minutes, nombre de personnes : 4 et score : 3.8
- La vraie tarte tropézienne : durée de préparation : 60 minutes, nombre de personnes : 6 et score : 3.9

On peut constater que les recettes retournéess respectent effectivement les critères entréss par l'utilisateur.

Wines/Pairing/

curl http://localhost:8080/Wines/Pairing/?recipe=flan_au_chocolat

La requête ci-dessus cherche des vins qui s'accord bien avec la recette flan au chocolat.
L'application nous retourne les vins suivants :

- Barolo DOCG Nebbiolo Cannubi Borgogno (75cl) 2013: "pairsWellWith":[{"terms":"cuisine italienne"},{"terms":"ragouts"},{"terms":"plats mijotes"},{"terms":"fromage"},{"terms":"chasse"},{"terms":"boeuf"}]
- Barbera d'Alba DOC Borgogno (75cl) 2019": "pairsWellWith":[{"terms":"cuisine italienne"},{"terms":"chasse"},{"terms":"boeuf"},{"terms":"fromage"}]

Ici il n'y pas vraiment de lien entre les pairwellswith et la recette recherchée. Il s'agit surement du fait que la recette n'a pas beaucoup de termes qui match avec les pairwells. Les seuls termes qui match sont surement des termes comme des adjectifs ou des déterminants. Ceci n'est pas très optimisé mais avec la version utilisant le modèle word2vec ce problème devrait être réglé.

/Recipes/Pairing

curl http://localhost:8080/Recipes/Pairing/?wine=lavaux

La requête cherche les recettes qui s'accorde bien avec les vins contenant le terme lavaux.

L'application nous retourne les recettes suivantes :

- Blanc de volaille au wok avec ses légumes sautes
- Saute de veau aux trompettes de la mort

Ici il est difficile d'extrapoler les résultats mais on peut constater que le vin lavaux à surement un pairwellswith qui contient le terme sauté car il apparait dans les 2 recettes.

Interface web

L'interface web est fonctionnelle et retourne les résultats escomptés. Les figures de la section Interface Web en page 27 sont tirées de l'application fonctionnelle. La communication avec le backend et le modèle est fonctionnelle.

Améliorations futures

Une des possibilités pour améliorer la rapidité du système et réduire la complexité de celui-ci est de transférer les poids (les vecteurs) du modèle word2vec directement en champ dans la base de données Elastic Search. Avec ceci réalisé, il serait possible de faire la distance cosine dans une query script. Exemple : https://www.ulam.io/blog/text-similarity-search-using-elasticsearch-and-python/
La fonctionnalité de suggestion de recettes depuis une recette aimée par l'utilisateur n'a pas pu être implémentée car il fallait un système pour connaitre les recettes qu'un utilisateur aime. Si le développement du projet continue, on peut imaginer une solution de compte permettant la suggestion de recette.
Un point mentionné mais pas implémenté est l'idée de fusionner plusieurs dataset par exemple pour mondovino et coop vin. Mais cette idée pourrait être portée plus loin et on peut imaginer chercher plus de recettes sur d'autre sites.
Aussi les query d'elastic search fonctionnent bien mais elles comportent un problème. Particulièrement dans la recherche par ingrédient. En effet, à cause de la fuzziness, certains termes sont mal interprétés par exemple poulet peut devenir moule. Si l'utilisateur ne cherche qu'un seul terme, alors il a des chances de tomber sur des recettes qui n'ont pas de lien avec sa demande de base.
Il serait intéressant dans ce cas de transformer les termes en identifiants uniques et avant la recherche, d'utiliser un modèle de machine pour matcher le mot avec l'identifiant (par exemple poulet = volaille).

Conclusion

Ce projet a été très instructif et ludique car le sujet était vraiment intéressant et utile pour la vie courante. Le système implémente toutes les fonctionnalités qui ont été établies dans le cahier des charges cependant des améliorations sont possibles comme vu précédemment. Les résultats obtenus sont convainquant et même si certains sont quelquefois étranges (remplacer du chocolat par du roquefort), la majorité des prédictions sont utilisables dans la réalité.

Table des figures