# Rapport Web Sémantique & Ingénierie des données

EL KATEB Sami, JEANNES Théo

Dans le cadre de notre projet, nous avons décidé de développer une application permettant aux utilisateurs de découvrir et de choisir des recettes de cuisine adaptées, en fonction de leur régime alimentaire et des ingrédients dont ils disposent. Pour cela, nous avons récupéré et structuré les données de plusieurs sources, afin de créer un ensemble cohérent, comportant toutes les informations nécessaires aux fonctionnalités souhaitées.

## Description et spécifications de l'application

Notre application propose une interface permettant aux utilisateurs de découvrir facilement de nouvelles recettes. Grâce à une fonction de recherche avancée, ils peuvent trouver des recettes par nom, ingrédients, ou en utilisant des filtres spécifiques. Cette fonctionnalité est particulièrement utile pour ceux qui souhaitent sélectionner des recettes en fonction de leurs préférences alimentaires, telles que des options végétariennes.

Lorsqu'une recette est sélectionnée, l'utilisateur est dirigé vers une page détaillée présentant les informations usuelles des recettes telles que le titre, la liste d'instructions et une image si celle-ci est disponible sur DBPedia. La page inclut également une liste complète des ingrédients, avec leurs quantités et unités respectives. Ces dernières peuvent être aisément converties entre les systèmes métrique et impérial.

Finalement, la page fournit des informations nutritionnelles adaptées à la recette et aux quantités d'ingrédients indiquées.

## Modélisation et contraintes sur les données

Pour représenter nos données, nous avons commencé par créer plusieurs vocabulaires et thésaurus. Nous avons d'abord créé le vocabulaire général du projet. Celui-ci permet de définir entre autres les classes *:Recipe*, *:Food*, *:Quantity* et *:Ingredient*, ainsi que les propriétés des recettes. Chaque recette est le sujet de plusieurs relations que nous avons définis dans ce vocabulaire. Les instructions d'une recette sont définies avec la relation *:instructions* et le nom de la recette est lié à la relation *:name*. Lorsque que cela est possible, la miniature d'une recette est renseignée avec la relation *:hasThumbnail* et un lien vers l'entrée Dbpedia de la recette est lié avec la relation *:hasDbpediaLink*.

Les ingrédients d'une recette sont définis par la relation *:hasIngredient*. Chaque ingrédient est représenté par un nœud anonyme. Ce nœud anonyme est relié à un nom par la relation *:name*, à un aliment lorsque cela est possible grâce à la relation *:food*. Il est également relié à la quantité et l'unité par les propriétés *:hasStandard-MeasurementUnit*, *:hasImperialMeasurementUnit* ou *:hasMetricMeasurementUnit*. Il est important de faire la distinction entre **un aliment** (*:Food*) qui est une substance qu'il possible de manger (lait, viande, pain) et un **ingrédient** (*:Ingredient*) qui fait partie d'une recette et qui est défini par une quantité, une unité et **un aliment** (*:Food*).

Dans le vocabulaire, nous avons défini la plupart des catégories de plats sous forme de classes. Ces classes nous permettent de typer nos recettes. Nous avons par exemple, la classe *:Cookie* qui est une sous-classe de la classe *:Dessert*. Ces données pourraient nous permettre de fournir plus d'informations sur les recettes ainsi que d'offrir plus de choix de recherche à l'utilisateur. De plus, nous avons aligné ce vocabulaire sur le vocabulaire fowl quand cela est possible.

Nous avons ensuite deux thésaurus, permettant de définir les unités de mesures et les aliments.

Le thésaurus des unités est divisé en trois concepts principaux et ses entités sont précédées du prefix *measurements:*. Les deux premiers groupes contiennent les unités du système métrique et les unités du système impérial (US). Les unités de ces deux groupes peuvent être converties d'un système vers l'autre, grâce à la relation *:hasUnitConversion*. L'objet de cette relation est un nœud anonyme, contenant l'unité vers laquelle convertir ainsi que le ratio de conversion. Le troisième groupe contient les unités dites standards n'ayant pas besoin d'être converties. Nous y retrouvons par exemples les tranches, les pincées et les cuillères à soupes. Ce thésaurus contient également trois quantités approximatives, lorsque la granularité n'est nécessaire (*measurements:Some*, *measurements:Few* et *measurements:Many*).

Le thésaurus des aliments contient quant à lui tous les aliments disponibles dans nos recettes. Ses entités sont précédées du prefix *food:* et il possède en concepts principaux toutes les grandes catégories d'aliments, comme la viande, le poisson, les fruits et les légumes. Les autres aliments sont liés à ces concepts principaux à l'aide de la relation *skos:broaderTransitive*. Ces relations nous permettent, une fois couplées à l'inférence OWL, de lier les recettes à des régimes alimentaires. Nous pouvons ainsi créer la classe *:RecipeWithAnimalProduct* qui regroupe les recettes ayant un aliment ayant pour concept plus large *food:animalProduct*. Nous pouvons ensuite en déduire les recettes veganes qui sont l'intersection des recettes et du complément de *:RecipeWithAnimalProduct*. Cependant, le raisonneur OWL RL ne réaliser pas l'inférence complementOf qui est prise en charge par les raisonneurs OWL DL. Nous avons donc créé une propriété *:complementaryRecipe* que nous utilisons pour ce cas d'usage. Nous l'utilisons pour générer les recettes complémentaires grâce à une requête SPARQL lancée au démarrage de notre serveur.

Une fois le vocabulaire et les thésaurus définis, nous avons pu utiliser OWL pour ajouter automatiquement plusieurs relations. Nous nous sommes par exemple servis d'un property axiom pour ajouter directement à chaque recette les aliments utilisés, et nous avons inféré l'inverse de cette propriété. Ceci nous permet de savoir dans quelle recette chaque aliment est utilisé et ainsi de simplifier les requêtes permettant de faire les recherches.

Pour s'assurer que les structures de données répondent à la forme souhaitée, nous avons également imposé des contraintes à l'aide de SHACL. Cela permet par exemple de s'assurer que les objets d'une relation *:hasUnitConversion* ont un ratio de conversion qui est un nombre décimal différent de 0.

Nous utilisons également SHACL pour vérifier la structure des recettes et des ingrédients. Par exemple, nous vérifions que les quantités soient soit des nombres décimaux non nuls soit l'une des valeurs correspondant aux quantités approximatives que nous avons déclarées dans le thésaurus des mesures. Finalement, cela nous permet de vérifier que l'unité liée à la quantité d'un ingrédient est une IRI qui correspond à une unité dans le thésaurus Measurements. Pour vérifier cette condition, nous pouvons utiliser la propriété sh:pattern qui permet de limiter les URI possibles aux URI qui désignent des unités.

## Traitement des données

Pour obtenir toutes les informations nécessaires à la réalisation de notre application, nous avons utilisé plusieurs sources de données et fait de nombreux traitements sur ces données, afin de les standardiser pour les rendre exploitables. Nous retrouvons une vue d'ensemble de ces traitements dans la figure 1.

### Extraction et Mise en forme depuis le CSV

Pour commencer, nous avons lifté les données d'un CSV contenant une liste de recettes en utilisant csv2rdf. Le CSV choisi décrit chaque recette avec une colonne pour le nom, une colonne pour la catégorie de recettes et une colonne avec les instructions. Les quantités, ingrédients et unités sont décrits dans les colonnes "UnitX", "IngredientX" et "QuantityX", ou X est un nombre entre 1 et 19. Les données de ces colonnes sont toutes des chaînes de caractères ou Null.

Une fois les données insérées dans un graph RDF avec csv2rdf, nous les avons transformées pour pouvoir les utiliser. Pour traiter ces données, nous avons utilisé Corese-command pour faire de multiples requêtes CONSTRUCT sur les données. La première chose a été de créer les nœuds anonymes désignant les ingrédients, qui regroupent un aliment, une quantité et une unité. Nous pouvons ensuite supprimer tous les ingrédients sans noms, ce qui évite les nombreux nœuds anonymes dûes aux colonnes Null dans le CSV.

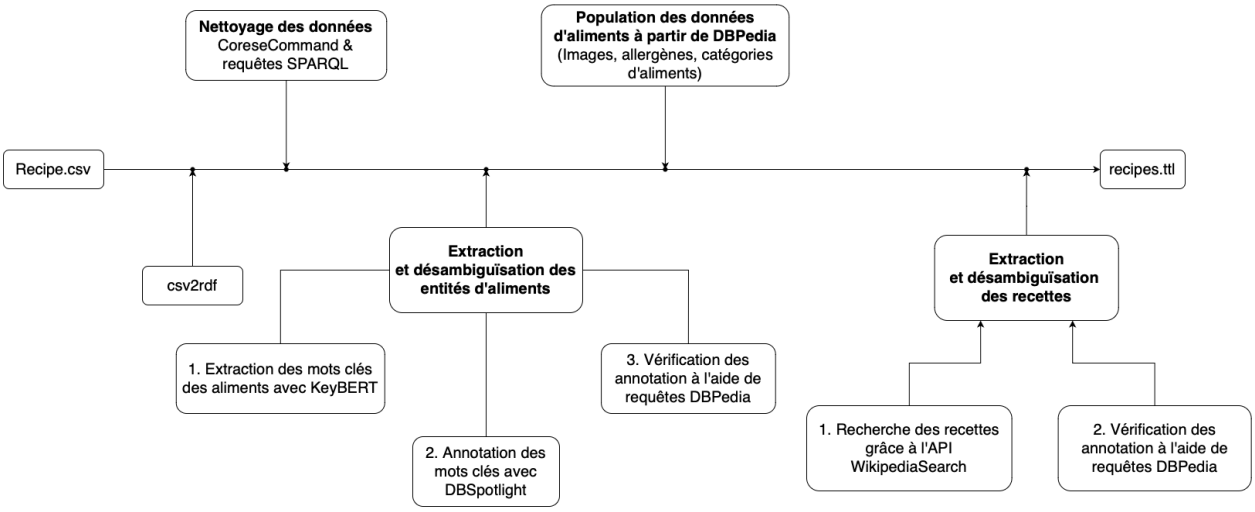


Figure 1: Schéma de la pipeline de traitement des données. Cette illustration détaille les étapes clés impliquées dans notre processus de traitement de données. Le processus commence par la conversion de fichiers CSV en format RDF. Ensuite, les données sont restructurées et nettoyées grâce à l'application de requêtes SPARQL. Ceci est suivi par l'extraction et la désambiguïsation d'entités à partir du texte. Finalement, les données sont enrichies à l'aide de DBPedia.

Une fois cela fait, il a été nécessaire de standardiser les unités et les quantités. Idéalement, toutes les quantités devraient être exprimées par un nombre décimal et toutes les unités devraient être un concept RDF défini dans le thésaurus Measurements. Pour ce faire, nous avons tout d'abord utilisé des expressions régulières pour extraire la quantité de l'unité lorsque cela était nécessaire. Par exemple l'unité "6 oz. pkg." contient à la fois l'unité, *(measurements:WeightOunce)* et la quantité (6). Nous extrayons donc cette quantité, que nous associons à la relation *:quantityFromUnit* de façon temporaire.

Nous avons ensuite utilisé des expressions régulières pour remplacer les chaînes de caractères par un maximum d'unités existantes dans le thésaurus Measurements.

Nous pouvons ensuite convertir les chaînes de caractères des quantités en unités. Pour cela, nous utilisons encore une fois des expressions régulières pour pouvoir récupérer les quantités. En effet, le CSV contient des quantités sous différents formats qu'il faut prendre en compte, comme "1.5", "1 ½" et "3/2".

Nous avons également introduit des quantités approximatives, qui sont *measurements:Few*, *measurements:Some* et *measurements:Many*, pour pouvoir mapper les quantités indéfinies dans le CSV comme "a few". Cela permettra de pouvoir les convertir du système métrique au système impérial et vice versa grâce à leur propriété *:hasQuantityValue*. Nous pouvons ainsi convertir *some ounces* en 3 ounces puis en 85 grammes.

Une fois les unités et les quantités définies, nous pouvons multiplier les quantités par le champ *:quantityFromUnit* récupéré précédemment et séparer les unités selon leur système de mesures. Cela permet d'identifier rapidement les unités impériales, ou standards, ce qui nous permet d'utiliser un CONSTRUCT pour ajouter facilement les quantités et unités converties dans un autre système de mesure, en utilisant la propriété *:conversionRatio*.

La dernière étape pour le CSV est de mapper la colonne Catégorie aux catégories de notre vocabulaire. Une fois cela fait, nous pouvons définir la catégorie comme le type des recettes.

**Extraction de mots-clés, désambiguïsation et linking**

Dans nos données, nous pouvons retrouver des ingrédients dont les aliments sont facilement identifiables comme "Tomato" mais aussi d'ingrédient plus difficile à identifier comme "broccoli florets, cut or broken". Pour pouvoir extraire correctement les entités depuis le texte, nous commençons par extraire les mots clés

des noms d'ingrédients à l'aide de KeyBERT. Pour cela, nous utilisons comme paramètre 1 à 3 n-grams car il est possible d'avoir des noms d'aliments en un mot comme "Potato" mais également des noms composés de plusieurs mots comme "Green onion". Une fois les différents n-grams récupérés, nous utilisons DBSpotlight pour retrouver les entités associées sur DBPedia. Finalement, nous vérifions que l'entité extraite est bien un aliment. Pour cela, nous effectuons une requête SPARQL à DBPedia pour savoir si l'entité est de type *(dbo:Food)*, si elle est l'ingrédient d'une recette *(dbo:mainIngredient)*, si elle possède un ingrédient ou si elle possède des informations nutritionnelles *(dbo:protein)*. Si l'entité extraite n'est pas un aliment, nous essayons le mot clef suivant fourni par KeyBERT. Cette méthode nous a permis d'ajouter un aliment pour tous les ingrédients que nous avons reconnus. Par la suite, nous avons effectué des requêtes à DBPedia pour obtenir des informations supplémentaires sur les ingrédients. Notamment, nous avons cherché à quelle catégorie d'aliments ceux-ci appartenaient (viande, légumes, fruits). Ces informations nous permettent par la suite d'attribuer chaque recette à un ou plusieurs régimes alimentaires.

Par la suite, nous avons souhaité retrouver nos recettes sur DBPedia lorsque celles-ci y sont présente. Cependant, dans ce cas, les résultats fournis par DBSpotlight nous ont semblé moins pertinents que les résultats fournis par la fonctionnalité de recherche de Wikipedia. Nous avons donc choisi d'utiliser l'API Wikipedia Search pour rechercher directement une page Wikipedia associée au nom de la recette. Nous pouvons ensuite effectuer une requête a DBPedia pour récupérer l'entité DBPedia associée à cette page. Si celle-ci est trouvée, nous lions notre recette à la recette dans DBpedia, et nous ajoutons à notre recette l'image miniature associée *(:hasThumbnail)*.

**Inférence sur les données**

Au démarrage du serveur, nous commençons par ajouter à notre graph la réciproque de notre propriété *:hasUnitConversion*. Puis nous convertissons nos unités à l'aide des ratios de conversion. Enfin nous ajoutons également les types de recettes complémentaires aux types définit grâce à OWL.

En effet, en utilisant les aliments que nous avons extraits des noms d'ingrédients, nous pouvons tirer avantage des classes définies avec OWL pour trouver les classes qui contiennent de la viande, des produits laitiers, des noix, des arachides et des produits d'origine animale.

En utilisant les compléments de ces recettes, grâce à *:complementaryRecipe*, nous pouvons facilement définir toutes les recettes qui ne comportent pas un ingrédient. Par exemple, en définissant les recettes qui contiennent de la viande et du poisson comme *:RecipeWithMeatOrFish*, nous pouvons utiliser le complément pour définir *:VegetarianRecipe*.

La propriété *owl:complementOf* aurait permis de faire le complément sans définir une nouvelle relation. Cependant, *owl:complementOf* n'existe que dans OWL DL. Nous avons donc créé cette relation, et nous avons ajouté l'inférence de cette relation à l'aide d'une requête INSERT sur notre graph. Cela permet d'étendre le fonctionnement de OWL pour prendre en charge un comportement qui n'aurait pas été possible sinon.

Définir ces catégories de recettes nous permet de proposer simplement des filtres à l'utilisateur.

**Requête fédérée**

Pour compléter les données affichées à l'utilisateur, nous avons utilisé à la fois un microservice SPARQL pour récupérer les données de l'API CalorieNinjas et un endpoint DBpedia.

Pour pouvoir afficher les informations nutritionnelles de la recette, nous avons fait un microservice qui utilise CalorieNinjas pour récupérer toutes les informations nutritionnelles des ingrédients en fonction de la quantité, et qui additionne les valeurs des ingrédients, pour avoir entre autres, la quantité de sucre ou le nombre de calories de la recette.

Pour faire appel à ce microservice de façon dynamique, lorsque l'utilisateur accède à une recette, nous construisons la query de la requête au service avec les quantités et le nom de chaque ingrédient. Nous concaténons ensuite cette query à l'URL du microservice. À travers cette requête, nous pouvons récupérer les informations de la recette depuis notre graphe, les informations nutritionnelles depuis le microservice

CalorieNinjas et un commentaire permettant de décrire la recette lorsque nous disposons du lien vers l'entité DBPedia.

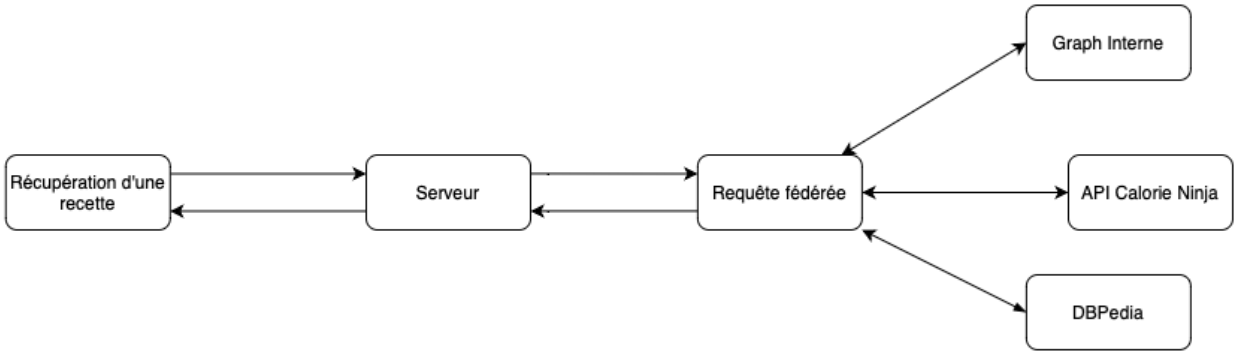


Figure 2: Diagramme de séquence pour une requête fédérée. Cette illustration représente les différentes étapes de la récupération d'informations relatives à une recette. Initialement, une requête est envoyée à notre serveur. Ce dernier procède ensuite à une requête fédérée vers le serveur Corese. Le serveur Corese interroge alors notre graphe de connaissances ainsi que deux sources externes : le microservice agissant comme un proxy pour l'API Calorie Ninja et la base de connaissances DBpedia.