An Advanced Recipe Recommendation System

Mahshid Jafar Tajrishi Leontina van Kampen
Radis Marios Toumpalidis Sanjana Somashekar

Group 7 - January 18th, 2026

1. Application domain and goals

Along with the continuous advancement of web technologies, food-related content on the web has expanded considerably, leading to a proliferation of platforms where users can access content tailored to their own specifications. Due to the unstructured representation of a website's content in web 2.0 applications users have to spend a lot of time searching for recipes or other content in order to find something similar to their own queries. Even though research communities have worked on creating data-driven or machine learning systems and tools that utilize NLP methodologies or semantic web technologies there is not a centralized service that can provide fast and reliable food information retrieval across multiple web sources. Such implementations include the work in [3], where the authors trained a neural recipe generation model on a 2 million recipe processed dataset to generate recipes based on user queries. Another state of the art implementation involves the development of the food data API [8] by the U.S. department of agriculture that offers several datasets about a wide range of foods along with dietary, nutritional and allergy information and can easily be integrated to semantic web applications.

In our implementation, we leveraged the capabilities of semantic web technologies to create a recipe recommendation system that produces semantically similar recipes based on user queries. Particularly, we created a unified knowledge graph from three different recipe sources we collected, resulting in a graph store that can retrieve recipes through an API that queries the knowledge graph. Such queries utilize SPARQL functionalities to retrieve recipes based on filters that concerns like dietary or allergy restrictions such as vegan or nut-free and/or nutritional constraints regarding calorie or protein intake.

2. Datasets Used

2.1. Dataset Collection and Cleaning

The main datasets for this project come from RecipeNLG [3], the Spoonacular API [6], and TheMealDB [7] (for recipes) and the USDA FoodData Central API [8] (for nutritional information). TheMealDB provides open-access recipe data, including ingredients, measurements, instructions, categories, and cuisines. From this source, we retrieved recipe categories, ingredient lists, random meals, and filtered search results, such as recipes by category (e.g., vegetarian or seafood) or by specific ingredients (e.g., salmon or chicken). The USDA dataset offers detailed nutritional information for common ingredients, including calories, protein, and fat content.

After collecting the data, we cleaned and processed it to prepare it for conversion into RDF. The cleaning process included handling timeouts and retries, as well as parsing the data. For example, ingredient names were normalized by removing qualifiers such as "fresh" or "chopped", and measurements were converted to grams, with limits applied to avoid extreme values.

2.1.1. RecipeNLG dataset

The recipeNLG [3] dataset was created on top of the Reciper1M+ dataset [4] along with web scraped data. Several modifications were made from its creators to accompany the training and deployment of a neural recipe generation model. It consists of more than 2 million recipes along with their required ingredients, links to recipes' web sources, instructions, and an author constructed column called NER (Named Entity recognizer) that was used to recognize food entities to enhance the understanding of their proposed model. Besides the fact that the dataset was used for machine learning training it could easily serve as a knowledge graph source for our recipe recommendation system due to its abundance of recipes, combined with detailed ingredient lists and instructions. In addition, it offers the opportunity to apply entity linkage with other recipe sources we collected and accommodate us in creating a unified knowledge graph.

2.1.2. Spoonacular Dataset

The Spoonacular API is another key data source for the knowledge graph. The Spoonacular API is a commercial recipe and food data service that provides structured access through its API. It offers comprehensive recipe data including ingredient lists with precise measurements, step-by-step cooking instructions, nutritional information and other information such as cooking time, serving size, and dietary classifications. Spoonacular enriches each recipe with dietary labels such as vegan, vegetarian, gluten-free, dairy-free, and ketogenic, making it particularly valuable for users with specific dietary requirements. The API also provides cuisine classifications and maintains links to original recipe sources for attribution. For this project, 155 recipes were retrieved from Spoonacular.

2.2. Dataset Transformation

After cleaning the data, the dataset was transformed into RDF using the Python library rdflib. The knowledge graph was constructed using several namespaces, including Schema.org for recipe and nutrition concepts, a custom EX namespace for the project specific entities, and the FOOD ontology for ingredient representation. And the core classes defined in the graph are Recipe, Ingredient, and Nutrition Information.

2.2.1. RecipeNLG to RDF transformation

The procedure of generating the recipeNLG RDF included creating a relational model for the recipes and ingredients and distinguishing them using auto increment identifiers. Given the established relations, then for each row of the recipe dataset several triples were added to the graph regarding the recipe's label, links and instructions. After processing a recipe, for each of the recipe's ingredients we distinguished between the lexical measurement units and quantities as such:

```
"1 1/2 cups of Flour" →
{
    "ingredient_text": "flour",
    "primary_unit": "cups",
    "secondary_unit": None,
    "primary_qty": 1,
    "secondary_qty": 1/2
}
```

and added all ingredient triples as unique entities with their derived information such as quantities, units, URIs and recipe order.

A string normalization process to remove special characters such as ",", ".", "_" was applied to ingredient and recipe names in order to create valid labels and URIs. Images in figure 1 display the turtle representation of recipes and ingredients from the recipeNLG dataset. Additionally in the process of converting the tabular recipes and ingredients datasets into RDF we utilized owl, xsd, rdfs, dcterms , and food namespaces.

Figure 1: Turtle representation of recipes and ingredients from recipeNLG dataset.

2.2.2. Spoonacular Data to RDF

The structured nature of the API responses simplified the transformation to RDF. The transformation process involved fetching data from the API endpoints and converting the JSON responses into RDF triples using the Python rdflib library. The API responses contained structured recipe objects with nested ingredient and nutrition information, which needed be separated and mapped to appropriate RDF properties. For each recipe, a unique URI was generated using the pattern `http://example.org/food/recipe/id`, where the ID corresponded to the Spoonacular recipe identifier. For ingredients, ingredient names were normalized to create valid URIs by converting to lowercase, removing special characters (commas, parentheses), and replacing spaces with underscores. For example, "Chicken Breast, boneless" became `http://example.org/food/ingredient/chicken_breast_boneless`. All recipes did not contain complete information, the transformation script used optional patterns and null checks to handle missing fields.
After the initial conversion, the vocabulary had to be modified to match the group's unified vocabulary established. This was necessary to enable seamless querying across all data sources without requiring UNION clauses for different property names. After vocabulary alignment, the JSON fields were mapped to RDF properties using the unified vocabulary.

2.3. Constructing a unified knowledge graph

In order to create a unified graph given the three separated graphs derived from the 3 different sources we used, we created a procedure divided in three steps that:

1. Merged the three turtle representations into one knowledge graph, without any modifications
2. Applied entity linkage between ingredients from all sources
3. Applied normalization to ensure all entities use the same namespaces in their representations (e.g. all recipes are instances of the http://data.lirmm.fr/ontologies/food#Recipe class)

This procedure allowed us to integrate the resulting graph with an API, without having to account for different sources and vocabularies in our queries also improving query execution times and generating simpler understandable queries.

Figure 2: Turtle representation of Spoonacular recipe after vocabulary alignment, showing recipe classification

3. Semantic Web Technologies

3.1. Ontology and Reasoning

The ontology was constructed by using two different tools: RDFlib to create all basic classes, data properties, and object property structures, and Protégé to add RDFS/OWL axioms for constructing and enriching the ontology. The domains, ranges, and characteristics of the key object properties were defined. As an example, the hasNutrition property was modeled as a functional object property from Ingredient to NutritionInformation, ensuring that each ingredient is linked to at most one nutrition profile. The ingredient usage pattern was established by representing the relationship:

`Recipe → IngredientUsage → Ingredient`

through the properties ingredientUsage and usesIngredient, with isIngredientOf as the inverse of both usesIngredient and hasIngredient.

4. Application Structure and Example results

4.1. Visualization and User Interface

The knowledge graph is visualized using NetworkX and Matplotlib through a Tkinter based interface. With this, users can adjust layout options, limit the number of displayed recipes, and interactively explore nodes. A command line interface manages the main processing steps, including data loading, testing, and visualization. Figure [3].

4.2. Query System

SPARQL is a standard query language made for the RDF data, and it is used to extract information from the recipe knowledge graph. In the knowledge graph there is a lot of data about recipes, ingredients, labels and nutrition. By using SPARQL the system can support various queries that allow filtering for different recipes, ingredients, labels and nutritional information. There are various queries incorporated in the system. Users could search for recipes containing or excluding certain ingredients in recipes, for example; peanuts or tofu. This can help users find fitting recipes according to their dietary restrictions or allergies. The query system can also count the amount of ingredients that have nutritional information, find ingredients that do not

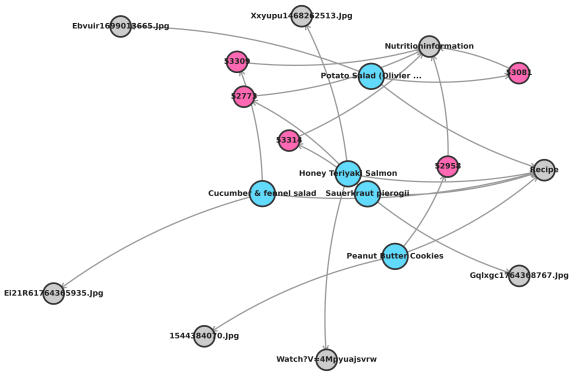

Figure 3: Visualization of the knowledge graph, where blue nodes represent recipes, pink nodes represent nutritional information, and gray nodes correspond to additional related details.

have nutritional information, or that have a missing label. This is important for checking the data quality in the knowledge graph. The system also checks ingredients that have more than one nutritional information and the most common ingredients. If the most common ingredient is found it can help in preperation with cooking for certain users, to be able to find which ingredients they should have. Other valuable queries that the system uses is ordering recipes based on their cuisine type, and being able to distinguish the different types of meat as classify them as such. Beef, chicken, lamb and pork will be classified as meat, while other ingredients will be classified as "vegetarian".

4.3. Web Application

The web interface was implemented as a FastAPI based client that accessed recipe data by issuing SPARQL queries to an external Apache Jena Fuseki endpoint and returning the returned results without performing local data storage or graph management.

4.3.1. API & Query Integration

The back-end of our recipe application consists of two services, a knowledge graph triple store and a simple python API system that executes queries. In particular, we deployed the standalone Apache Jena Fuseki store [2] which offers several functionalities on managing knowledge graphs and connecting other services to it, and we created a FastAPI [5] instance that connects to the Fuseki store and enables the execution of SPARQL queries. In order to make our system os-agnostic we leveraged Docker containerization to package both the Fuseki store and FastAPI application into reproducible services. In detail, after importing our graph to Fuseki store the FastAPI container connects to Fuseki's SPARQL endpoint through environment-based configuration, allowing the API to execute queries via HTTP requests.

5. Techniques Used

5.1. Graph Embeddings For Recipe Similarity

Knowledge graph embeddings were implemented using the PyKEEN [1] framework with the RotatE model to generate vector representations for entities in the recipe knowledge graph. The RotatE model was trained with an embedding dimension of 64 for 50 epochs using a batch size of 256. The embeddings were trained offline on entity to entity relationships derived from the

graph structure and stored for later use by the application. Cosine similarity between recipe embeddings was computed at runtime to support a "similar recipes" feature in the web interface. Through this way, semantic similarity was captured, allowing recipes with similar ingredients, cuisines, or dietary properties to be represented by closely related vectors. This enables users to discover alternative recipes, explore related cuisines, and identify new recipes aligned with their preferences.

6. Limitations And Future Work

- Same ingredient appears with different names across sources (e.g., "tofu", "firm tofu") making exact matching difficult
- Searching by ingredient returns imprecise results due to inconsistent labeling across sources (e.g., searching "chicken" returns non chicken recipes)
- Different data sources (RecipesNLG, MealDB, Spoonacular) originally used different vocabularies, and while alignment was performed, some inconsistencies are still present.
- Complex SPARQL queries on 1.2M triples can be slow
- The embeddings are computed previously, so newly added recipes would require retraining the model.
- The similarity is based purely on the graph structure and does not account for user preferences or ratings.
- With 64-dimensional embeddings trained for 50 epochs on a subset of relations, the model may not capture all nuances of recipe similarity. Given more time and computational resources, we could experiment with higher dimensions and more training epochs.
- Expand external knowledge base linking using automated NLP techniques
- Develop advanced search with ingredient substitution recommendations

References

[1] Mehdi Ali et al. Pykeen 1.0: A python library for training and evaluating knowledge graph embeddings. *JMLR*, 2021.

[2] Apache Software Foundation. Apache jena fuseki, 2025. Accessed: 2026-01-17.

[3] Michał Bień, Michał Gilski, Martyna Maciejewska, Wojciech Taisner, Dawid Wisniewski, and Agnieszka Lawrynowicz. RecipeNLG: A cooking recipes dataset for semi-structured text generation. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 22–28, Dublin, Ireland, December 2020. Association for Computational Linguistics.

[4] Javier Marin, Aritro Biswas, Ferda Ofli, Nicholas Hynes, Amaia Salvador, Yusuf Aytar, Ingmar Weber, and Antonio Torralba. Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2019.

[5] Sebastián Ramírez. Fastapi: Modern, fast (high-performance) web framework for building apis with python, 2024. Accessed: 2026-01-17.

[6] Spoonacular. Spoonacular api. Accessed: 14 January 2026.

[7] TheMealDB. Themealdb api. Accessed: 14 January 2026.

[8] U.S. Department of Agriculture. Usda fooddata central api. Accessed: 14 January 2026.

A. Project Details

A.1. Team Contributions

Mahshid Jafar Tajrishi	Data collection/cleaning, Converting Data to RDF, Ontology, Creating SPARQL queries, Tkinter Visualization
Leontina van Kampen	Creating SPARQL queries
Radis Marios Toumpalidis	RecipeNLG data source collection, Converting to RDF, linking with other graphs, Graph Database & API system
Sanjana Somashekar	Spoonacular data source collection, Converting to RDF, linking with External knowledge Base, Graph embeddings & web Interface

A.2. Generative AI Usage Statement

During the completion of this project, we utilized generative AI tools (e.g., ChatGPT) in a limited capacity to assist in specific areas:

- In programming, these tools were employed for tasks such as bug fixing, troubleshooting, and investigating technical challenges. We did not use AI for large-scale code generation, and all core functionalities and logical designs were independently developed by the team.
- In report writing, Grammarly was used as a language enhancement tool to refine grammar and improve clarity. The report's content, structure, and main ideas were entirely conceived and finalized by the authors.

This statement is included to ensure transparency regarding the use of generative AI tools throughout the project.