

# AI60008: KNOWLEDGE MODELING AND SEMANTIC WEB TECHNOLOGIES



## Ingredient-Based Recipe Finder

Swapnil Shekhar

20MA20059

Vaishnavi Malwade

19CH30035

Varun Balakrishna

19CH30036

Vatsal Venkatkrishna

19CH30037

# Introduction

The problem of not knowing what to cook is as old as humanity. These days we're left clueless about what to cook with a limited set of ingredients while maybe factoring in dietary restrictions. At the same time, we live in the information age and have access to unlimited data at our fingertips. This data consists of a well-documented and detailed list of recipes that one can use to discover new ways to satisfy dietary requirements. So, the project's goal is to allow users to efficiently and quickly gain access to this knowledge base to solve day-to-day problems.

## Problem Statement and Objectives

The problem statement essentially boils down to the following – “How can a user find recipes that fit their dietary needs by inputting a certain set of ingredients that they want/don't want.” This problem statement can be broken down into a series of subtasks that have to be solved. They are as follows:

- Building a front-end interface for the user to input requirements, i.e., ingredients.
- Sending the data to the backend for further processing, where the input data is initially preprocessed.
- Getting a list of all ingredients available on Wikidata along with their URI.
- Getting the URI of the closest match of entered ingredients and the available list.
- Building the SPARQL query to retrieve a list of recipes that satisfy all criteria.
- Formatting and displaying the result on the front end.

## Methodology

The front end is built using Anvil, a free Python-based drag-and-drop web app builder. The back end is written in Python and is run as a Google Colab Notebook. The following libraries were used to write the program:

- Anvil.server is a library to connect securely to the Anvil server, allowing the Anvil app to call functions in your project.
- SPARQLWrapper is a simple Python wrapper around a SPARQL service to remotely execute your queries. It helps create the query invocation and, possibly, convert the result into a more manageable format.

Our goal is to return all dishes having one or more required ingredients, and no forbidden ingredients. For this purpose, we query the Wikidata knowledge base using the SPARQLWrapper module in Python which makes inputting a list of required and forbidden ingredients as from the user convenient. We first query a list of all foods and their respective Wikidata QIDs and store the result in a dictionary. Further, we convert the inputs from the user to the closest match available in our dictionary of ingredients (Eg: If American cheese is not present in our dictionary, we map it to the QID of cheese, and so on). We then use the UNION and MINUS constructs offered by SPARQL for the required and forbidden ingredients filter respectively. It is important to note that the difference between Figure 1a) and 1b). Figure 1a) will only count a dish if it has all required

ingredients present, and similarly disqualify a dish only if all forbidden ingredients are present. Thus, we make use of stacked UNION and MINUS blocks as in Figure 1b)

```
SELECT ?ing
WHERE
{
  ?food wdt:is_instance_of wd:food;
        wdt:has_part(s) wd:required_1;
        wdt:has_part(s) wd:required_2.
  MINUS
  {
    ?food wdt:has_part(s) wd:forbidden_1;
          wdt:has_part(s) wd:forbidden_2.
  }
}
```

Figure 1a) Query without stacked constructs

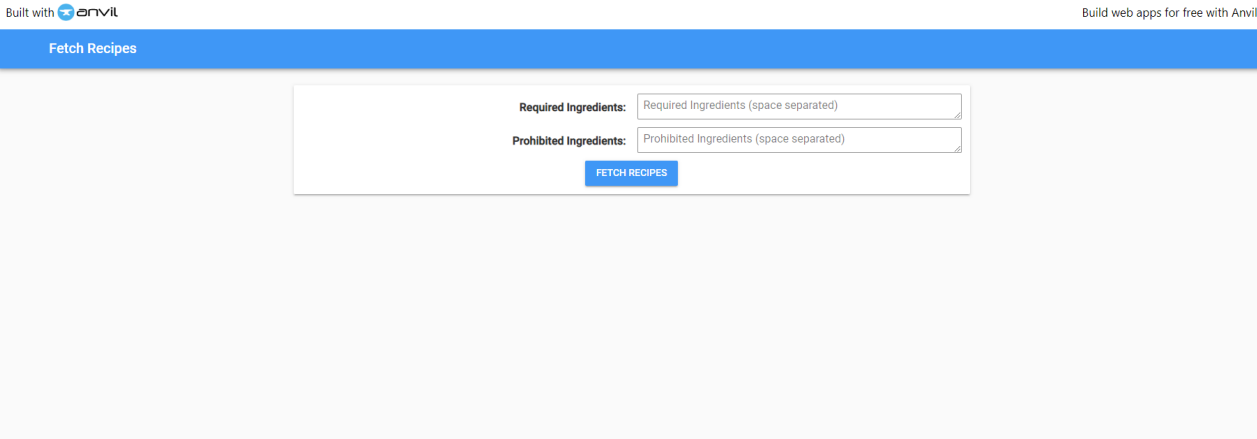
```
SELECT ?ing
WHERE
{
  ?food wdt:is_instance_of wd:food.
  {wdt:has_part(s) wd:required_1.}
  UNION
  {wdt:has_part(s) wd:required_2.}
  MINUS{?food wdt:has_part(s) wd:forbidden_1.}
  MINUS{?food wdt:has_part(s) wd:forbidden_2.}
}
```

Figure 1b) Query with stacked constructs

We use Python list comprehensions to extend this functionality to any number of required and forbidden ingredients and pass the final query to a SPARQLWrapper instance, which runs our query on wikidata and returns the results in a JSON format. We then use a Pandas DataFrame to make the results more readable.

## Deployment

We created a website using Anvil, a Python-based tool for creating and deploying web applications. The website allows users to input desired and prohibited ingredients and retrieve recipe results from a Google Colab notebook. The website has a simple and aesthetically pleasing design, with two input text boxes and a button to prompt the result.



The screenshot shows a web application interface built with Anvil. At the top, there's a blue header bar with the text "Fetch Recipes". Below this, there's a white container with a light gray border. Inside the container, there are two input fields. The first is labeled "Required Ingredients:" and has a placeholder text "Required Ingredients (space separated)". The second is labeled "Prohibited Ingredients:" and has a placeholder text "Prohibited Ingredients (space separated)". Below these two input fields is a blue button with the text "FETCH RECIPES".

Here is a demo to show how the website looks after it fetches the data based on input prompts

Required Ingredients: "Tomato Cheese Lettuce Bread"

Prohibited Ingredients: "Beef Ham"

Required Ingredients:

Tomato Cheese Lettuce Bread

Prohibited Ingredients:

beef ham

FETCH RECIPES

|   | Food        | Ingredients |
|---|-------------|-------------|
| 0 | Masala puri | Puri Bread  |
| 1 | Sevpuri     | Puri Bread  |
| 2 | puri bhaji  | Puri Bread  |

## Links

GitHub Repo: <https://github.com/vatsal-kr/Recipe-KMST>

YouTube: <https://youtu.be/zqxtqqTaLRk>

Website: <https://kmst-project-iitkgp.anvil.app>

## References

1. Foundations of Semantic Web Technologies, Pascal Hitzler, Markus Krotzsch, Sabastian Rudolph, CRC Press, 2009
2. [Wikidata: Crowd-sourced Structured Knowledge by Wikimedia Foundation - https://www.wikidata.org/wiki/Wikidata:Main\\_Page](https://www.wikidata.org/wiki/Wikidata:Main_Page)
3. <https://query.wikidata.org/>
4. <https://anvil.works/docs/overview>
5. <https://plaban.github.io/kmst-spring-2023/>
6. <https://sparqlwrapper.readthedocs.io/en/latest/>