PROJECT

KNOWLEDGE ENGINEERING & BUSINESS INTELLIGENCE

Personalized Menu

Msc. Computer Science

WORKED BY:

Angjelina Bahushi

Elva Panariti

ACCEPTED BY:

Prof.Dr. Knut Hinkelmann

Dr.Emanuele Laurenzi

TABLE OF CONTENTS

1.Introduction

This project explores the development of an intelligent meal recommendation system designed to address the limitations of traditional digital restaurant menus. Conventional digital menus simply display a list of dishes, offering no support for tailoring suggestions to individual guests with specific dietary needs, allergies, or nutritional preferences. To move beyond this static approach, the project adopts semantic technologies and knowledge-based methods to create a system capable of understanding and interpreting detailed culinary and customer information.

Central to this work is a comprehensive semantic model of Italian cuisine that represents meals, ingredients, and guest profiles in a structured and meaningful way. Through the use of ontologies, automated reasoning, and formal knowledge structures, the system can interpret complex relationships within the domain and support more informed and contextualized meal suggestions. The business process is also refined using BPMN 2.0, enabling the recommendation logic to integrate seamlessly into the operational workflow of a restaurant, ensuring clarity, automation, and consistency in how meal suggestions are produced.

1.1 Objectives

The goal of this project is to develop an intelligent Knowledge Base capable of generating personalized meal recommendations based on a customer's dietary preferences, restrictions, and nutritional needs. To accomplish this, the project integrates several semantic and rule-based technologies, each supporting a specific layer of the system:

- **DMN & BPMN (Decision Modeling and Business Process Integration):** Used to model structured decision logic and incorporate a dedicated Preference Processing Task into the overall business workflow.

- **Prolog:** Implements rule-based reasoning to validate dietary constraints, perform inference, and evaluate meal suitability.

- **Protégé:**
  Supports ontology engineering by defining the conceptual structure of meals, ingredients, guest profiles, and dietary rules.

SWRL & SHACL:

- SWRL enables automatic inference of meal suitability, while SHACL ensures data quality and validates knowledge graph structure.

- **SPARQL & Jena Fuseki:**
  Used for querying the ontology and retrieving personalized, constraint-aware meal suggestions.

All project files, implementation details, and documentation are available on GitHub: https://github.com/elvapanariti/Kebi_Project

1.2 Ingredients and Meals Overview

The **ingredients table** captures the essential properties of each food component using the format:

ingredient(Name, Dairy, Gluten, Peanuts, Vegetarian, Kcal)

For every ingredient, the model specifies whether it contains common allergens (dairy, gluten, peanuts), whether it is suitable for vegetarians, and its caloric value. This structure enables detailed filtering and analysis, such as identifying allergen-free foods or calculating nutritional values.

The **meals table** represents complete dishes using the structure:

meal(Name, Ingredients, CalorieLevel, IsVegetarian, AvailableTime)

Each meal entry lists its ingredient composition, a calorie classification (low, medium, or high), whether the dish is vegetarian, and the typical time when it is served (breakfast, lunch, dinner, or dessert).

Together, these models form a clear and flexible representation of food data. They support essential tasks such as allergen detection, nutritional evaluation, and the generation of personalized meal recommendations.

1.2.1 Ingredients Table

Ingredient	Dairy	Gluten	Peanuts	Vegetarian	Kcal
Tomato	No	No	No	Yes	18
Mozzarella	Yes	No	No	Yes	280
Chicken	No	No	No	No	239
Tofu	No	No	No	Yes	76
Bread	No	Yes	No	Yes	265
Peanut Butter	No	No	Yes	Yes	588
Rice	No	No	No	Yes	130
Broccoli	No	No	No	Yes	34
Milk	Yes	No	No	Yes	42
Olive Oil	No	No	No	Yes	884
Egg	No	No	No	Yes	155
Beef	No	No	No	No	250
Ham	No	No	No	No	145

Salmon	No	No	No	No	208
Zucchini	No	No	No	Yes	17
Eggplant	No	No	No	Yes	25
Pasta	No	Yes	No	Yes	131
Parmesan	Yes	No	No	Yes	431
Truffle	No	No	No	Yes	284
Mushrooms	No	No	No	Yes	22
Mascarpone	Yes	No	No	Yes	435
Rice Flour	No	No	No	Yes	366
Lettuce	No	No	No	Yes	15
Carrot	No	No	No	Yes	41
Potato	No	No	No	Yes	77
Pecorino	Yes	No	No	Yes	387
Bacon	No	No	No	No	541
Duck	No	No	No	No	337
Mayonnaise	No	No	No	Yes	680

Onion	No	No	No	Yes	40
Lemon	No	No	No	Yes	29
Avocado	No	No	No	Yes	160
Coffee	No	No	No	Yes	2
Sugar	No	No	No	Yes	387
Strawberry	No	No	No	Yes	32
Peach	No	No	No	Yes	39
Water	No	No	No	Yes	0

1.2.2 Meals Table

Meal Name	**Ingredients**	**Calorie Level**	**Vegetarian**	**Available Time**
Caprese Salad	Tomato, Mozzarella, Olive Oil	Low	Yes	Lunch
Chicken Rice Bowl	Chicken, Rice, Broccoli	Medium	No	Lunch
Tofu Stir Fry	Tofu, Broccoli, Olive Oil	Low	Yes	Dinner

Peanut Butter Toast	Bread, Peanut Butter	High	Yes	Breakfast
Grilled Chicken	Chicken, Olive Oil	Medium	No	Dinner
Vegetable Rice	Rice, Broccoli, Olive Oil	Low	Yes	Lunch
Mediterranean Plate	Tomato, Tofu, Olive Oil, Broccoli	Low	Yes	Dinner
Pasta alla Carbonara	Pasta, Egg, Pecorino, Bacon, Olive Oil	High	No	Dinner
Pasta al Pomodoro	Pasta, Tomato, Olive Oil, Parmesan	Low	Yes	Lunch
Eggplant Parmigiana	Eggplant, Tomato, Mozzarella, Parmesan, Olive Oil	High	Yes	Dinner
Salmon Plate	Salmon, Lemon, Olive Oil, Zucchini	Medium	No	Dinner
Grilled Vegetables	Zucchini, Eggplant, Tomato, Olive Oil	Low	Yes	Lunch
Garden Salad	Lettuce, Tomato, Carrot, Olive Oil, Lemon	Low	Yes	Lunch
Chicken Salad	Chicken, Lettuce, Tomato, Olive Oil	Medium	No	Lunch

Pizza Margherita	Pasta, Tomato, Mozzarella, Olive Oil	Medium	Yes	Dinner
Pizza Prosciutto	Pasta, Tomato, Mozzarella, Ham, Olive Oil	High	No	Dinner
Pizza Truffle	Pasta, Mozzarella, Truffle, Mushrooms, Olive Oil	High	Yes	Dinner
Duck Ragu Pasta	Pasta, Duck, Tomato, Olive Oil, Onion	High	No	Dinner
Ravioli Ricotta Spinach	Pasta, Ricotta, Parmesan, Olive Oil	Medium	Yes	Lunch
Beef Steak Plate	Beef, Olive Oil, Potato	High	No	Dinner
Mixed Grill	Chicken, Beef, Olive Oil, Lemon	High	No	Dinner
Vegetarian Arancini	Rice, Mozzarella, Tomato, Olive Oil	Medium	Yes	Lunch
Tiramisu	Coffee, Mascarpone, Sugar	High	Yes	Dessert
Avocado Toast	Bread, Avocado, Olive Oil	Medium	Yes	Breakfast

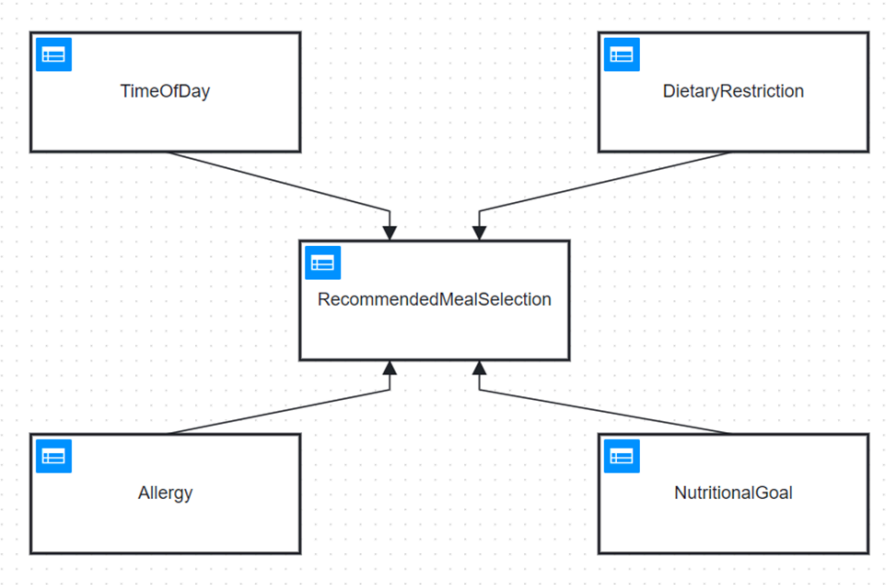
2. Decision Tables (Camunda)

Camunda is a platform known for automating and improving business processes, through BPMN(Business Process Model and Notation) and DMN (Decision Model and Notation) for decision logic. Camunda DMN was used to manage the decision-making process for selecting meals. After creating the decision models, they can be run directly on the Camunda platform.

Decision tables are used to represent complex decision-making logic in a clear and structured manner. They serve as a key tool for selecting the most appropriate meal for a guest based on multiple criteria. The decision process considers factors such as **time of day**, **dietary restrictions**, **allergies**, and **nutritional goals**, all of which influence which meals are suitable and recommended.

The diagram illustrates the relationship between these inputs and the **RecommendedMealSelection** decision. Each input contributes to filtering and ranking potential meal options:

- **TimeOfDay**
- **DietaryRestriction**
- **Allergy**
- **NutritionalGoal**



2.1 Time of Day Decision Table

This table shows which meals are appropriate depending on the time of day. It ensures that breakfast, lunch, dinner, or dessert options are selected correctly. By mapping meal types to specific time ranges, it helps the system avoid recommending meals that unsuitable for a particular time.

TimeOfDay | **Hit policy:** First

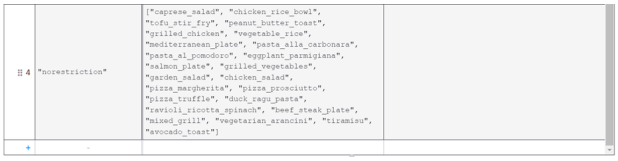
	When	Then	
	timeOfDay "breakfast","lunch","dinner","des...	timeFilteredMeals string	Annotations
1	"breakfast"	["peanut_butter_toast", "avocado_toast"]	
2	"lunch"	["caprese_salad", "chicken_rice_bowl", "vegetable_rice", "pasta_al_pomodoro", "grilled_vegetables", "garden_salad", "chicken_salad", "ravioli_ricotta_spinach", "vegetarian_arancini"]	
3	"dinner"	["tofu_stir_fry", "grilled_chicken", "mediterranean_plate", "pasta_alla_carbonara", "eggplant_parmigiana", "salmon_plate", "pizza_margherita", "pizza_prosciutto", "pizza_truffle", "duck_ragu_pasta", "beef_steak_plate", "mixed_grill"]	
4	"dessert"	"tiramisu"	
+			

2.2 Dietary Restrictions Decision Table

This table filters meal options based on a guest's dietary preferences. It ensures that meals comply with vegetarian, vegan, carnivore, or if the guest does not have any dietary restrictions or preferences.

DietaryRestriction | **Hit policy:** First

	When	Then	
	dietaryRestriction "vegan","vegetarian","carnivore",...	restrictionFilteredMeals string	Annotations
1	"vegan"	["tofu_stir_fry", "vegetable_rice", "mediterranean_plate", "grilled_vegetables", "garden_salad", "avocado_toast"]	
2	"vegetarian"	["caprese_salad", "tofu_stir_fry", "peanut_butter_toast", "vegetable_rice", "mediterranean_plate", "pasta_al_pomodoro", "eggplant_parmigiana", "grilled_vegetables", "garden_salad", "pizza_margherita", "pizza_truffle", "ravioli_ricotta_spinach", "vegetarian_arancini", "tiramisu", "avocado_toast"]	
3	"carnivore"	["chicken_rice_bowl", "grilled_chicken", "pasta_alla_carbonara", "salmon_plate", "chicken_salad", "pizza_prosciutto", "duck_ragu_pasta", "beef_steak_plate", "mixed_grill"]	



4	"norestriction"	["caprese_salad", "chicken_rice_bowl", "tofu_stir_fry", "peanut_butter_toast", "grilled_chicken", "vegetable_rice", "mediterranean_plate", "pasta_alla_carbonara", "pasta_al_pomodoro", "eggplant_parmigiana", "salmon_plate", "grilled_vegetables", "garden_salad", "chicken_salad", "pizza_margherita", "pizza_prosciutto", "pizza_truffle", "duck_ragu_pasta", "ravioli_ricotta_spinach", "beef_steak_plate", "mixed_grill", "vegetarian_arancini", "tiramisu", "avocado_toast"]	
+			

2.3 Allergy Decision Table

This table identifies and excludes meals containing allergens. It protects guests by preventing recommendations with ingredients like dairy, gluten, peanuts, or other allergens. The system cross-checks all meal options against the guest's allergy information.

Allergy | Hit policy: Collect

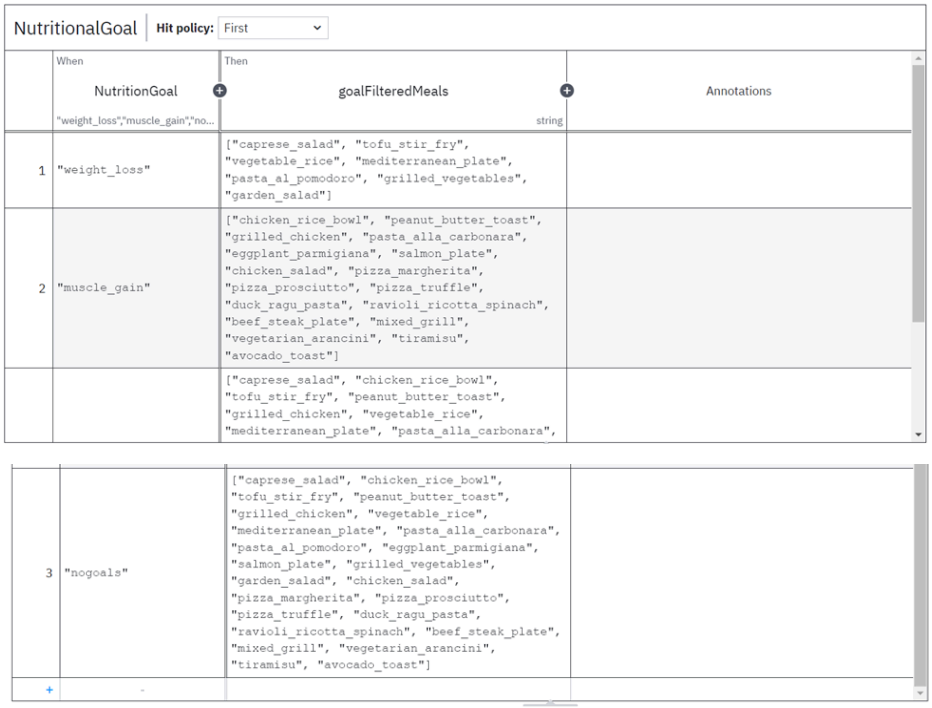
	When: Allergy "dairy","gluten","peanuts","noall...	Then: SafeMeals string	Annotations
1	"dairy"	["tomato", "chicken_rice_bowl", "tofu_stir_fry", "peanut_butter_toast", "grilled_chicken", "vegetable_rice", "mediterranean_plate", "salmon_plate", "grilled_vegetables", "garden_salad", "chicken_salad", "duck_ragu_pasta", "beef_steak_plate", "mixed_grill", "avocado_toast"]	
2	"gluten"	["caprese_salad", "chicken_rice_bowl", "tofu_stir_fry", "grilled_chicken", "vegetable_rice", "mediterranean_plate", "eggplant_parmigiana", "salmon_plate", "grilled_vegetables", "garden_salad", "chicken_salad", "beef_steak_plate", "mixed_grill", "vegetarian_arancini", "tiramisu"]	
		["caprese_salad", "chicken_rice_bowl",	

Allergy | Hit policy: Collect

	When: Allergy "dairy","gluten","peanuts","noall...	Then: SafeMeals string	Annotations
3	"peanuts"	"tofu_stir_fry", "grilled_chicken", "vegetable_rice", "mediterranean_plate", "pasta_alla_carbonara", "pasta_al_pomodoro", "eggplant_parmigiana", "salmon_plate", "grilled_vegetables", "garden_salad", "chicken_salad", "pizza_margherita", "pizza_prosciutto", "pizza_truffle", "duck_ragu_pasta", "ravioli_ricotta_spinach", "beef_steak_plate", "mixed_grill", "vegetarian_arancini", "tiramisu", "avocado_toast"]	
4	"noallergy"	["caprese_salad", "chicken_rice_bowl", "tofu_stir_fry", "peanut_butter_toast", "grilled_chicken", "vegetable_rice", "mediterranean_plate", "pasta_alla_carbonara", "pasta_al_pomodoro", "eggplant_parmigiana", "salmon_plate", "grilled_vegetables", "garden_salad",	

2.4 Nutritional Goals Decision Table

This table guides meal selection according to individual nutritional goals. It prioritizes meals based on guest preferences like: weight loss, muscle gain or no nutritional goal. This ensures recommendations are both personalized and goal-oriented.

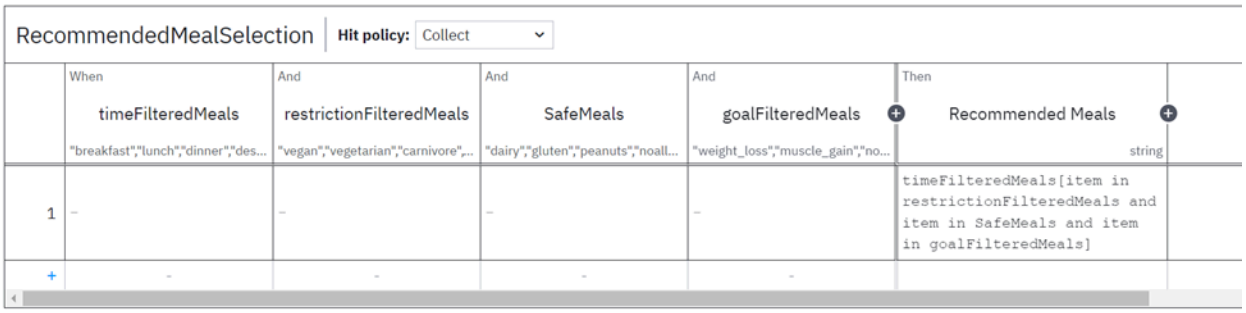


NutritionalGoal | Hit policy: First

	When	Then	
	NutritionGoal "weight_loss","muscle_gain","no...	goalFilteredMeals string	Annotations
1	"weight_loss"	["caprese_salad", "tofu_stir_fry", "vegetable_rice", "mediterranean_plate", "pasta_al_pomodoro", "grilled_vegetables", "garden_salad"]	
2	"muscle_gain"	["chicken_rice_bowl", "peanut_butter_toast", "grilled_chicken", "pasta_alla_carbonara", "eggplant_parmigiana", "salmon_plate", "chicken_salad", "pizza_margherita", "pizza_prosciutto", "pizza_truffle", "duck_ragu_pasta", "ravioli_ricotta_spinach", "beef_steak_plate", "mixed_grill", "vegetarian_arancini", "tiramisu", "avocado_toast"]	
		["caprese_salad", "chicken_rice_bowl", "tofu_stir_fry", "peanut_butter_toast", "grilled_chicken", "vegetable_rice", "mediterranean_plate", "pasta_alla_carbonara",	
3	"nogoals"	["caprese_salad", "chicken_rice_bowl", "tofu_stir_fry", "peanut_butter_toast", "grilled_chicken", "vegetable_rice", "mediterranean_plate", "pasta_alla_carbonara", "pasta_al_pomodoro", "eggplant_parmigiana", "salmon_plate", "grilled_vegetables", "garden_salad", "chicken_salad", "pizza_margherita", "pizza_prosciutto", "pizza_truffle", "duck_ragu_pasta", "ravioli_ricotta_spinach", "beef_steak_plate", "mixed_grill", "vegetarian_arancini", "tiramisu", "avocado_toast"]	
+			

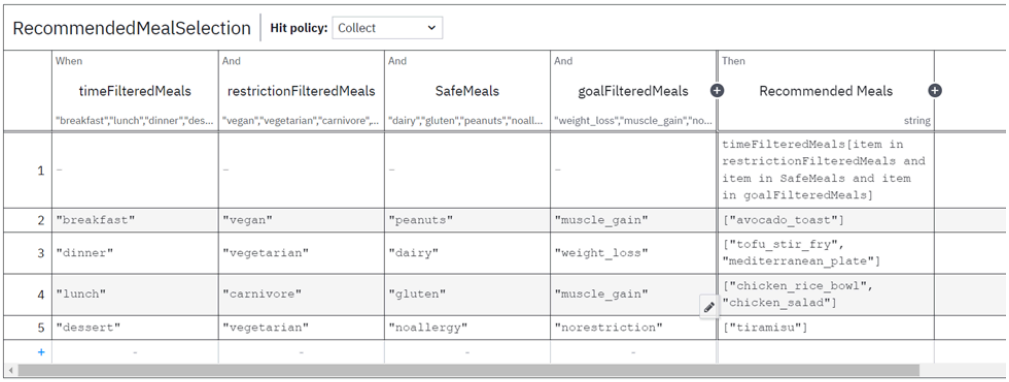
2.5 The Main Decision: RecommendedMealSelection

This table is the final component that combines the results from the four filtering decisions into a single, conclusive list of recommendations.



The expression performs an **intersection** of the four lists. It starts with the timeFilteredMeals list and keeps an item (a meal name) **only if** that same item is present (in) the other three lists (restrictionFilteredMeals, SafeMeals, and goalFilteredMeals). This guarantees that every recommended meal satisfies all four user constraints simultaneously.

2.6 Testing



RecommendedMealSelection | Hit policy: Collect

	When	And	And	And	Then
	timeFilteredMeals "breakfast","lunch","dinner","des...	restrictionFilteredMeals "vegan","vegetarian","carnivore",...	SafeMeals "dairy","gluten","peanuts","noall...	goalFilteredMeals "weight_loss","muscle_gain","no...	Recommended Meals string
1	-	-	-	-	timeFilteredMeals[item in restrictionFilteredMeals and item in SafeMeals and item in goalFilteredMeals]
2	"breakfast"	"vegan"	"peanuts"	"muscle_gain"	["avocado_toast"]
3	"dinner"	"vegetarian"	"dairy"	"weight_loss"	["tofu_stir_fry", "mediterranean_plate"]
4	"lunch"	"carnivore"	"gluten"	"muscle_gain"	["chicken_rice_bowl", "chicken_salad"]
5	"dessert"	"vegetarian"	"noallergy"	"norestriction"	["tiramisu"]
+	-	-	-	-	

I tried multiple times to run the simulation and test the model in Camunda but I could not achieve that. My version did not support testing and I was unable to get an automatic result. However, I tried to write some results hypothetically on what the system would recommend the guests with different preferences.

A: A guest is vegan and has an allergy of peanuts. His goal is muscle gain and he wants to eat something for breakfast.

B: A guest has come during dinner time with a weight loss goal and has an allergy of dairy products. He is vegetarian.

C: Another guest prefers meat dishes with no gluten for muscle gain during lunch time.

D: If another guest comes to our restaurant and wants a dessert, but he is vegetarian.

3. Prolog

```
/* Helper rules */
contains_allergen(Meal, lactose) :-
    meal(Meal, Ingredients, _, _, _),
    member(I, Ingredients),
    ingredient(I, true, _, _, _, _).

contains_allergen(Meal, gluten) :-
    meal(Meal, Ingredients, _, _, _),
    member(I, Ingredients),
    ingredient(I, _, true, _, _, _).

contains_allergen(Meal, peanuts) :-
    meal(Meal, Ingredients, _, _, _),
    member(I, Ingredients),
    ingredient(I, _, _, true, _, _).


suitable_for(Meal, Preferences) :-
    meal(Meal, _, _, _, _),
    \+ (member(lactose_free, Preferences), contains_allergen(Meal, lactose)),
    \+ (member(gluten_free, Preferences), contains_allergen(Meal, gluten)),
    \+ (member(peanut_free, Preferences), contains_allergen(Meal, peanuts)),
    ( \+ member(vegetarian, Preferences)
      ; (meal(Meal, _, _, true, _)) ).


recommend_meals(Preferences, CalorieLevel, Meals) :-
    findall(Meal,
        ( meal(Meal, _, CalorieLevel, _, _),
          suitable_for(Meal, Preferences)
        ),
        Meals).


recommend_meals(Preferences, CalorieLevel, Time, Meals) :-
    findall(Meal,
        ( meal(Meal, _, CalorieLevel, _, Time),
          suitable_for(Meal, Preferences)
        ),
        Meals).
```

3.1 Prolog Module Description

This Prolog module implements a rule-based recommender system for personalized meal selection in an Italian restaurant.

The knowledge base contains meals, their ingredients, and allergen information. The rules determine whether a meal is suitable for a customer based on dietary preferences and allergy restrictions.

The predicate **contains_allergen/2** identifies whether a meal includes lactose, gluten, or peanuts by checking the ingredient list.
The predicate **suitable_for/2** evaluates whether a meal fits the customer's profile, ensuring it excludes allergens when required and supports dietary needs such as vegetarian options.
Finally, the predicates **recommend_meals/3** and **recommend_meals/4** generate personalized meal recommendations by filtering meals according to preferences, calorie level, and optionally preparation time.

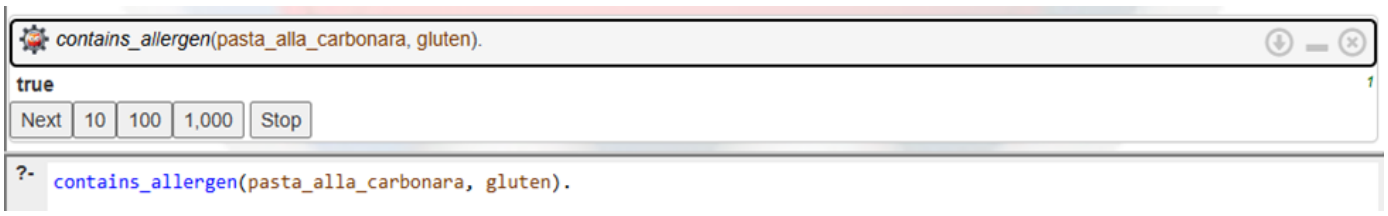
Overall, this Prolog component demonstrates how logical reasoning can automatically match restaurant menu items to customer dietary constraints using declarative rules.

3.2 Swish (Prolog Executer)

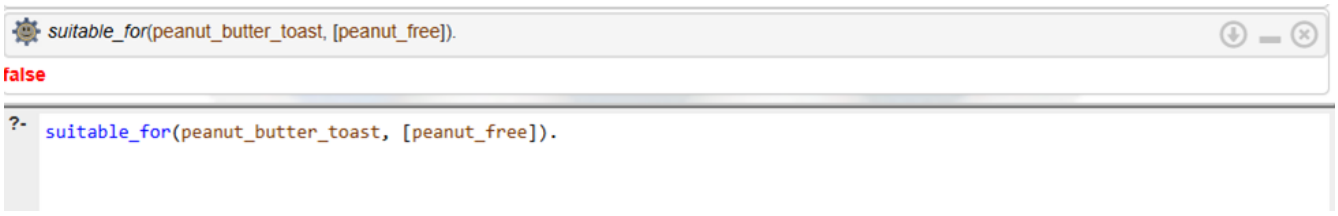
The code was tested and validated using **SWISH**, an online Prolog environment, where several queries were executed to ensure that the rules behave correctly and return appropriate meal suggestions according to different user profiles.

Below I have attached screenshots from SWI-Prolog (using SWISH online interface) showing the queries I executed to test different Prolog rules. The results are visible and match the expected outputs based on the dataset and the rule definitions I created.

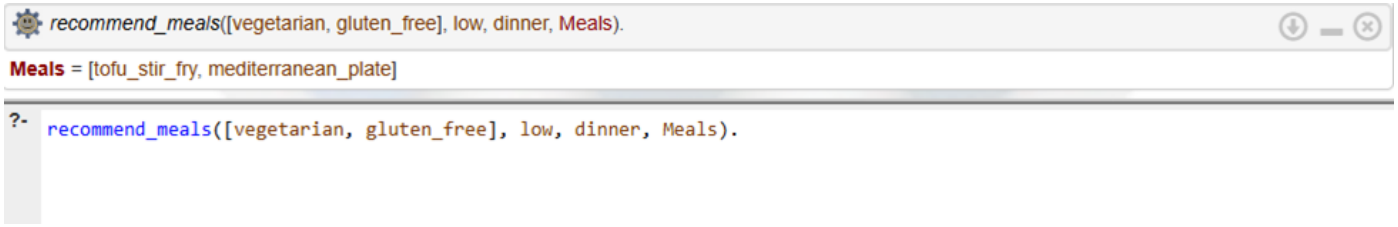
- **Check whether *pasta_alla_carbonara* contains gluten.**



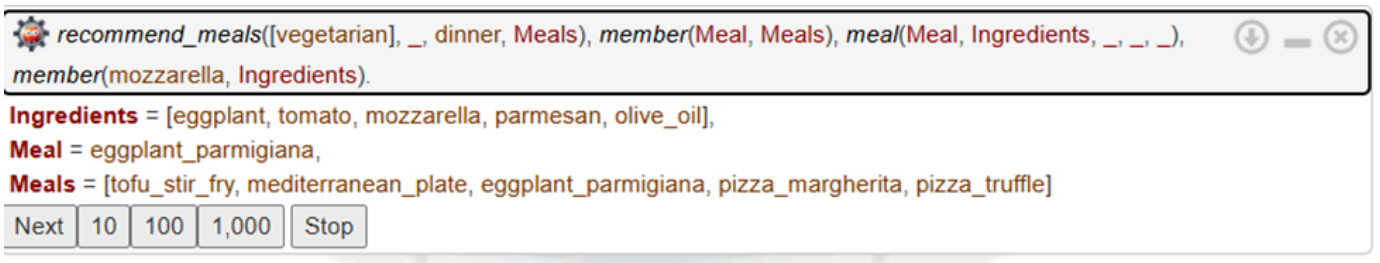
- **Check whether *peanut_butter_toast* is peanut-free**



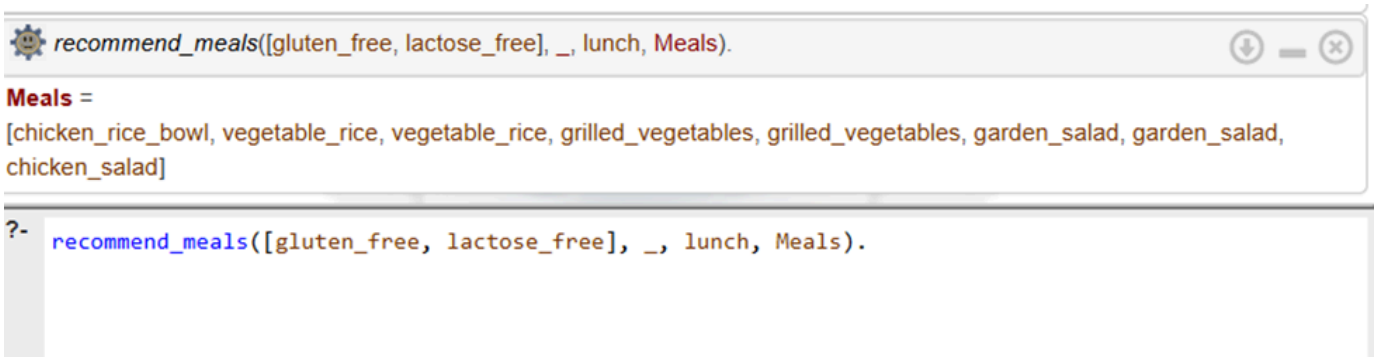
- **Dinner meals that are vegetarian and gluten-free.**



- **All meals that contain mozzarella and are suitable for dinner.**

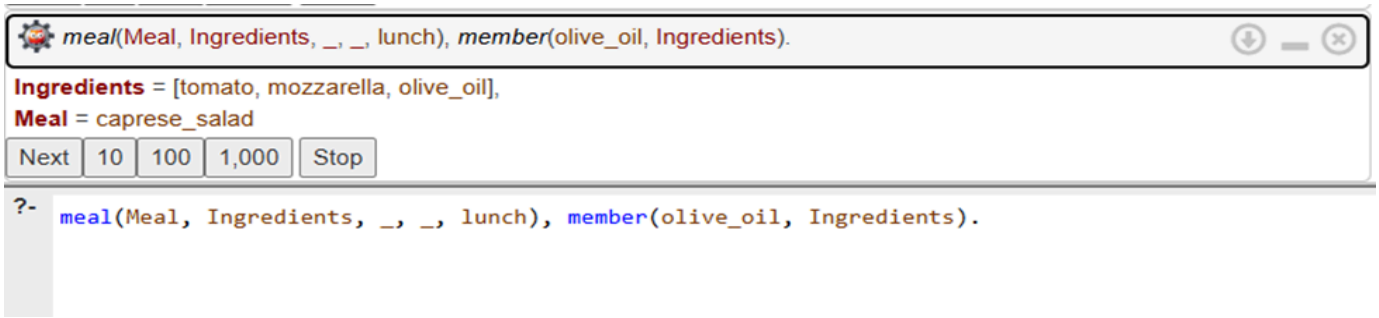


- **Lunch meals that are gluten-free and lactose-free.**

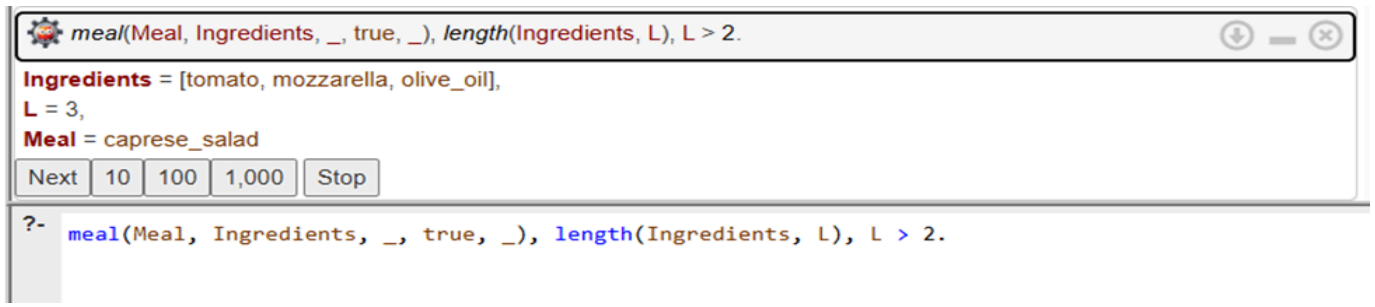


- **The most calorie-rich meals (over 400 kcal) that are vegetarian**

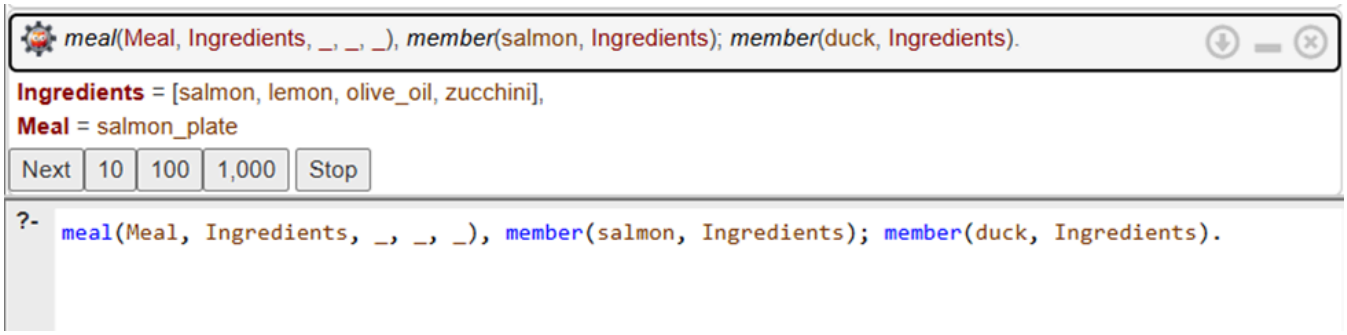
- **Find all the meals that contain olive oil and can be for lunch.**



- **Meals with more than 2 ingredients that are vegetarian.**

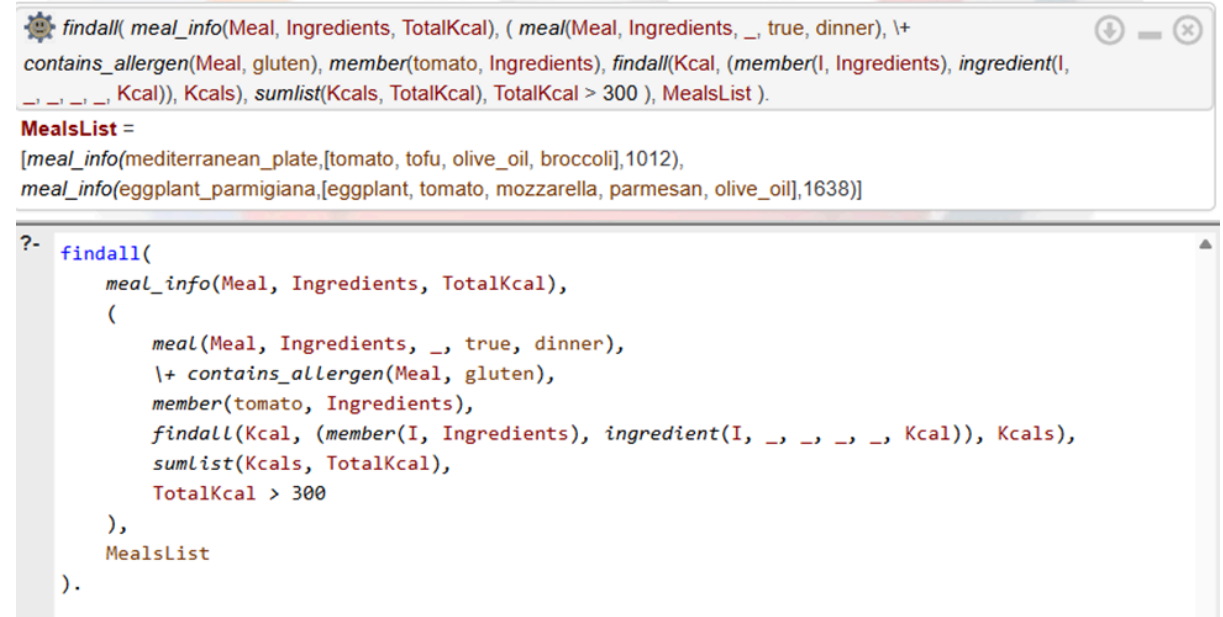


- **Meals that contain either salmon or duck.**



- **Dinner foods with olive oil and more than 300 kcal**

- **List of meals for dinner that are vegetarian, gluten-free, contain tomato, and have more than 300 kcal**



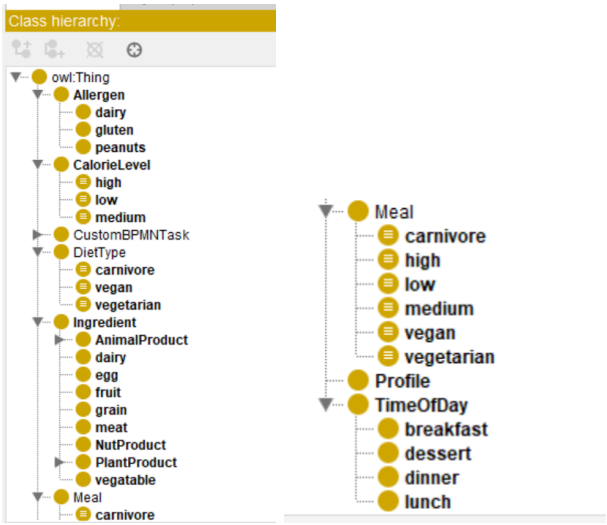
4. Knowledge Graph/ Ontology

4.1 Protégé

Protégé made it easier for me to develop a clear and consistent ontology. It allowed me to create classes, relationships between classes(object properties) and attributes (data properties), apply SWRL rules to express exclusion logic, and also export the ontology in suitable formats for integration with other tools like GraphDB.

Displayed in the picture below, are the classes and the subclasses I designed to capture the essential elements needed for meal classification and suitability checking.I organized the domain into classes such as **Meal**, **Ingredient**, **DietType**, **Allergen**, **TimeOfDay**, and **Profile**.

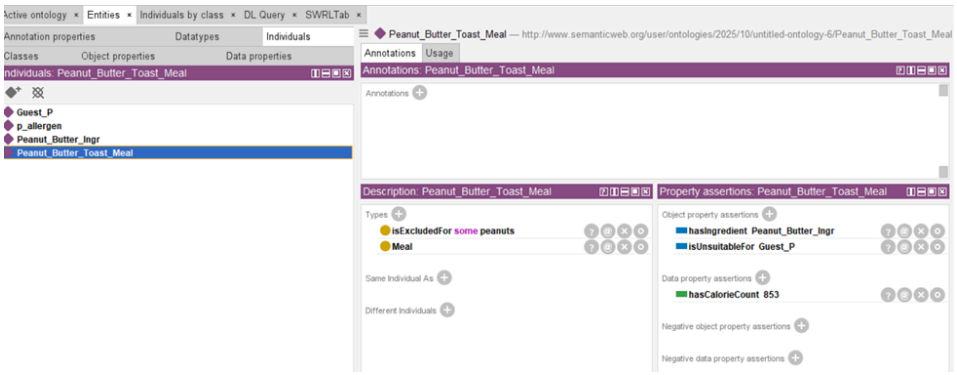
By structuring these concepts and defining how they relate to each other, I created a knowledge model that allows the system to reason about allergies, dietary needs, ingredient types, and meal characteristics.



4.2 Examples Implementation

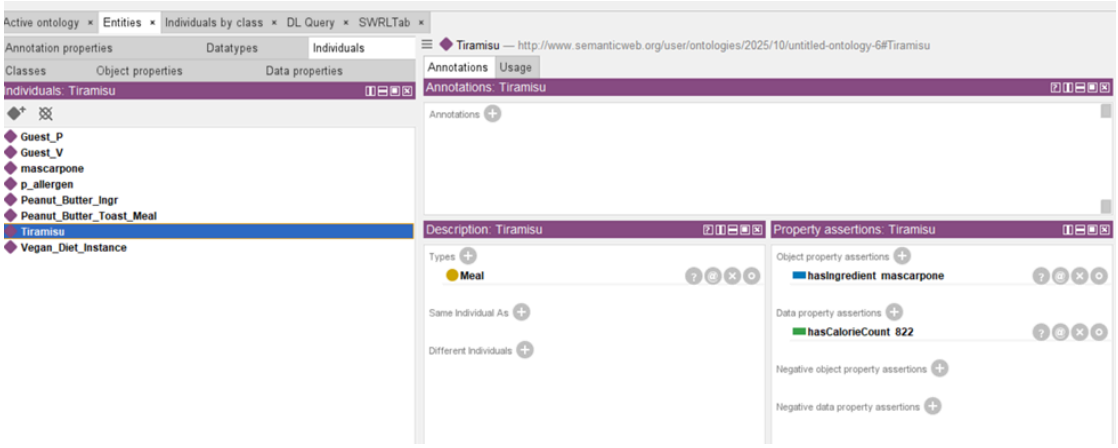
1. Allergy Exclusion

This test confirmed that the ontology correctly blocks meals containing allergens. A guest with a peanut allergy was linked to a peanut-based ingredient. When the reasoning process was run, the system correctly marked the peanut-containing meal as unsuitable for that guest.



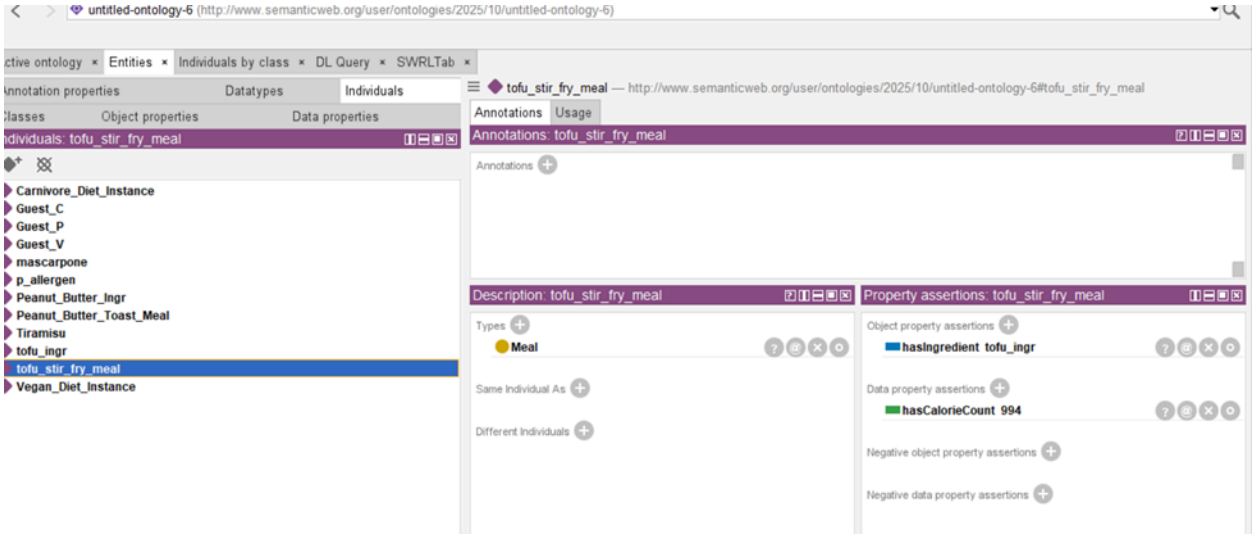
2. Dietary Exclusion

A vegan guest was associated with a non-vegan ingredient (mascarpone). The conflict was detected and flagged the Tiramisu meal as unsuitable for the vegan guest.



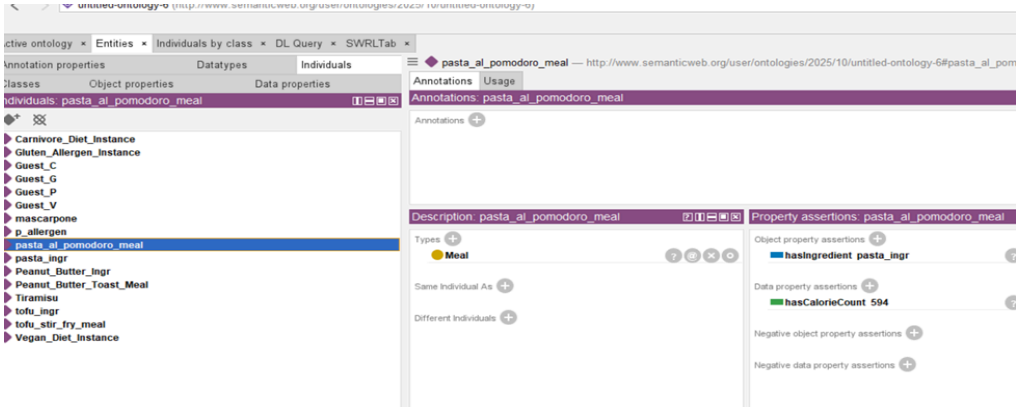
3. Dietary Exclusion (Carnivore Preference)

A carnivore guest was linked to plant-based ingredients. The tofu-based meal was successfully inferred as unsuitable, showing that non-restrictive preferences are also supported.



4. Safety Exclusion (Allergy)

This test validated gluten-related safety logic. A guest with a gluten allergy was connected to a gluten-containing ingredient within a meal. The engine flagged the pasta meal as unsuitable.



4.3 SPARQL Queries

For this part, I worked with SPARQL queries in GraphDB to pull information from the knowledge graph. Basically, I needed to ask the system questions like "show me all the meals" or "which guests have allergies" and SPARQL is the language that lets you do that.

I ran a bunch of different queries to make sure everything was connected properly - checking that meals had ingredients, guests had preferences, and the relationships between everything made sense. Most of the queries followed the same pattern, just asking about different things (meals vs guests vs ingredients).

This step was important because it proved that storing everything as a knowledge graph actually works - I could easily find connections between data that would be harder to track in a regular database. It gave me confidence that the system could handle the meal recommendation logic I built.

I will also share the query that provides the final result of the project, which displays the meals chosen by the client. To make the queries easier to understand, I used prefixes . Pre-fixes are shorthand for long URIs (Uniform Resource Identifiers), making the queries more readable by allowing us to use short, simple names instead of the full URLs.

- Here are some of the queries I created and tested, using different rules to check whether our menu is functioning properly. (For each query below you can find the result after running it).

- **Meals and their corresponding ingredients.**

```
PREFIX : <http://www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/>

SELECT ?mealName ?ingredientName WHERE {
    ?meal a :Meal .
    ?meal :hasIngredient ?ingredient .
    BIND(STRAFTER(STR(?meal), "/") AS ?mealName)
    BIND(STRAFTER(STR(?ingredient), "/") AS ?ingredientName)
}
ORDER BY ?mealName
```

	mealName	ingredientName
1	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Peanut_Butter_Toast_Meal"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Peanut_Butter_Ingr"
2	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Tiramisu"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/mascarpone"
3	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/pasta_al_pomodoro_meal"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/pasta_ingr"
4	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/tofu_stir_fry_meal"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/tofu_ingr"

- **This query finds meals and the guests they are unsuitable for.**

```
PREFIX : <http://www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/>

SELECT ?mealName ?guestName WHERE {
    ?meal :isUnsuitableFor ?guest .
    BIND(STRAFTER(STR(?meal), "/") AS ?mealName)
    BIND(STRAFTER(STR(?guest), "/") AS ?guestName)
}
ORDER BY ?guestName
```

	mealName	guestName
1	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/tofu_stir_fry_meal"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_C"
2	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/pasta_al_pomodoro_meal"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_G"
3	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Peanut_Butter_Toast_Meal"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_P"
4	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Tiramisu"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_V"
5	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_T"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/caprese_salad"

- **Meals that are suitable for a specific guest.**

```
PREFIX : <http://www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/>

SELECT ?mealName ?guestName WHERE {
    ?meal :isSuitableFor ?guest .
    BIND(STRAFTER(STR(?meal), "/") AS ?mealName)
    BIND(STRAFTER(STR(?guest), "/") AS ?guestName)
}
ORDER BY ?guestName
```

	mealName	guestName
1	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/caprese_salad"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_T"
2	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_P"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Peanut_Butter_Toast_Meal"
3	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_V"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Tiramisu"
4	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_G"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/pasta_al_pomodoro_meal"
5	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_C"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/tofu_stir_fry_meal"

- **Guest profiles and their dietary preferences or allergies.**

```
PREFIX : <http://www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/>

SELECT ?guestName ?preferenceType ?preferenceName WHERE {
    ?guest a :Profile .

    {
        ?guest :hasDietaryNeed ?pref .
        BIND("Dietary Need" AS ?preferenceType)
    }
    UNION
    {
        ?guest :hasAllergy ?pref .
        BIND("Allergy" AS ?preferenceType)
    }
```

Press Alt+Enter to autoco

	guestName	preferenceType	preferenceName
1	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_C"	"Dietary Need"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Carnivore_Diet_Instance"
2	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_G"	"Allergy"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Gluten_Allergen_Instance"
3	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_P"	"Allergy"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/p_allergen"
4	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_V"	"Dietary Need"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Vegan_Diet_Instance"

- **Ingredients and the specific dietary restrictions or groups they are excluded for.**

```
PREFIX : <http://www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/>

SELECT ?ingredientName ?excludedFor WHERE {
  ?ingredient :isExcludedFor ?restriction .
  BIND(STRAFTER(STR(?ingredient), "/") AS ?ingredientName)
  BIND(STRAFTER(STR(?restriction), "/") AS ?excludedFor)
}
ORDER BY ?ingredientName
```

	ingredientName	excludedFor
1	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Peanut_Butter_Ingr"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/p_allergen"
2	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/mascarpone"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Vegan_Diet_Instance"
3	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/pasta_ingr"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Gluten_Allergen_Instance"
4	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/tofu_ingr"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Carnivore_Diet_Instance"

- **Meals and the specific times of day they are served.**

```
PREFIX : <http://www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/>

SELECT ?mealName ?timeName WHERE {
  ?meal a :Meal .
  ?meal :servedAt ?time .
  BIND(STRAFTER(STR(?meal), "/") AS ?mealName)
  BIND(STRAFTER(STR(?time), "/") AS ?timeName)
}
ORDER BY ?timeName
```

	mealName	timeName
1	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/caprese_salad"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Lunch_Instance"

- **Available meals (with calorie count) that are suitable for a specific guest.**

```
PREFIX : <http://www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/>

SELECT DISTINCT ?mealName ?calories WHERE {
  VALUES ?guestURI { <http://www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_V> }
  ?meal a :Meal .
  OPTIONAL { ?meal :hasCalorieCount ?calories }

  FILTER NOT EXISTS { ?meal :isUnsuitableFor ?guestURI }

  BIND(STRAFTER(STR(?meal), "/") AS ?mealName)
}
ORDER BY ?calories
```

Press Alt+Enter to autocomplete

	mealName	calories
1	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Carnivore_Diet_Instance"	
2	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_C"	
3	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_G"	
4	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_T"	
5	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/caprese_salad"	
6	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_V"	
7	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Vegan_Diet_Instance"	
8	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_P"	
9	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/pasta_al_pomodoro_meal"	"594"^^xsd:integer
10	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Peanut_Butter_Toast_Meal"	"853"^^xsd:integer
11	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/tofu_stir_fry_meal"	"994"^^xsd:integer

- **Count of meals grouped by diet type (vegan, vegetarian, or carnivore).**

```
PREFIX : <http://www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/>

SELECT ?dietType (COUNT(DISTINCT ?meal) AS ?mealCount) WHERE {
  ?meal a :Meal .
  ?meal a ?type .

  FILTER(?type IN (:vegan, :vegetarian, :carnivore))

  BIND(STRAFTER(STR(?type), "/") AS ?dietType)
}
GROUP BY ?dietType
ORDER BY DESC(?mealCount)
```

	dietType	mealCount
1	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/vegan"	"1"^^xsd:integer
2	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/carnivore"	"1"^^xsd:integer

- **Meals that are unsuitable for a specific guest, along with the problematic ingredient and the reason (allergy or dietary need).**

```
PREFIX : <http://www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/>

SELECT ?mealName ?guestName ?problematicIngredient ?reason WHERE {
  ?meal :isUnsuitableFor ?guest .
  ?meal :hasIngredient ?ingredient .

  {
    ?guest :hasAllergy ?allergen .
    ?ingredient :isExcludedFor ?allergen .
    BIND(STRAFTER(STR(?allergen), "/") AS ?reason)
  }
  UNION
  {
    ?guest :hasDietaryNeed ?diet .
    ?ingredient :isExcludedFor ?diet .
    BIND(STRAFTER(STR(?diet), "/") AS ?reason)
  }

  BIND(STRAFTER(STR(?meal), "/") AS ?mealName)
  BIND(STRAFTER(STR(?guest), "/") AS ?guestName)
  BIND(STRAFTER(STR(?ingredient), "/") AS ?problematicIngredient)
}
ORDER BY ?guestName
```

	mealName	guestName	problematicIngredient	reason
1	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/tofu_stir_fry_meal"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_C"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/tofu_ingr"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Carnivore_Diet_Instance"
2	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/pasta_al_pomodoro_meal"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_G"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/pasta_ingr"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Gluten_Allergen_Instance"
3	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Peanut_Butter_Toast_Meal"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_P"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Peanut_Butter_Ingr"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/p_allergen"
4	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Tiramisu"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_V"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/mascarpone"	"/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Vegan_Diet_Instance"

These queries were a great way to showcase the power of SPARQL and GraphDB in our project. By running them, I confirmed that our knowledge graph can handle complex requests, from simple data retrieval to advanced relationship analysis.

Specifically, I used:

- Simple Lookups and Joins to find related information (like ingredients and serving times).
- Set Operators (UNION) and Aggregation (COUNT/GROUP BY) to calculate metrics (like meal counts per diet).
- Filtering (FILTER NOT EXISTS) to ensure that the results were relevant and suitable for specific profiles.
- Data Cleaning (BIND/STRAFTER) to make the results readable.

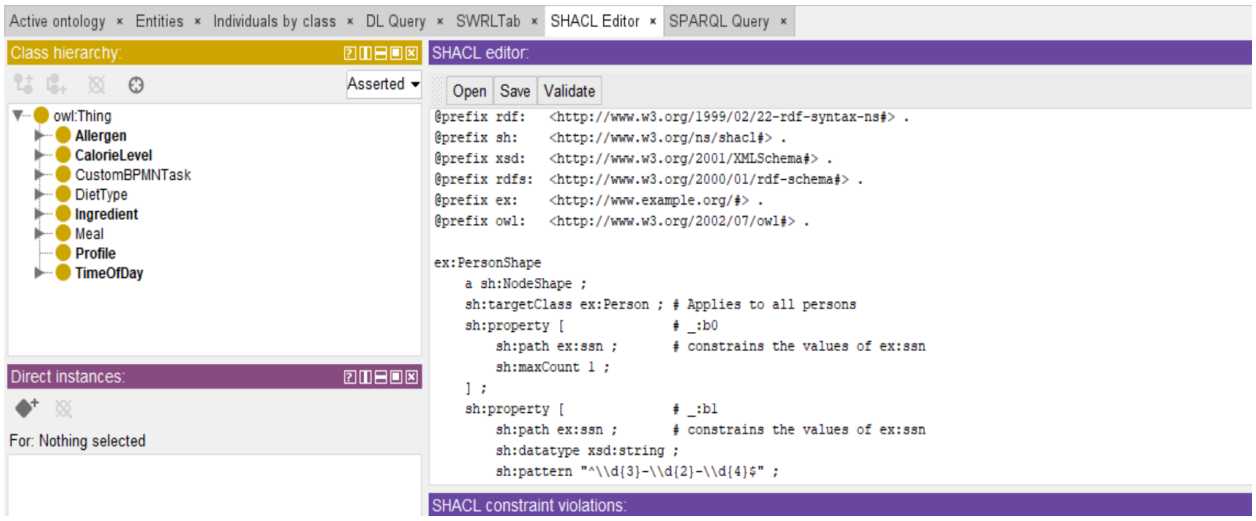
The successful execution and clear results of these queries confirm that our ontology is working well and provides a solid foundation for building applications that need to make smart, data-driven decisions.

4.4 SHACL Shapes

After creating the ontology and SPARQL queries, I needed a way to make sure the data stays consistent and follows the business rules. That's where SHACL comes in - it's basically a quality control system for the knowledge graph.

I created validation shapes that automatically check things like: every meal must have ingredients, every guest profile needs at least one preference (allergy or dietary need), and meals marked as vegetarian can't contain meat. SHACL also validates that calorie information is present and that recommended meals don't contain allergens for guests with allergies.

The main advantage is that these validations run automatically whenever data is added to Protege. If something violates a rule - like trying to add a meal without ingredients or recommending a peanut dish to someone with a peanut allergy - SHACL catches it immediately and shows an error message. This prevents bad data from breaking the recommendation system and ensures food safety constraints are always respected.



4.5 SWRL Rules

I created several SWRL rules that infer new relationships from the existing data.

1. Allergy Exclusion Rule

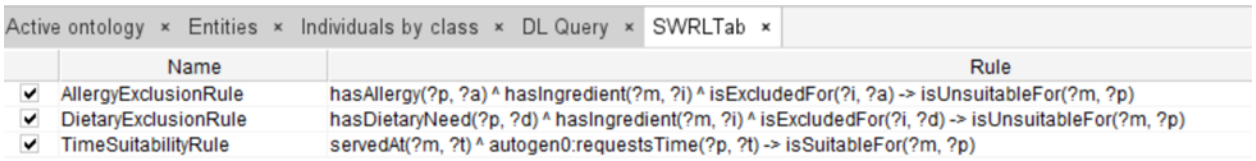
This rule allows me to automatically identify when a meal should be marked as unsuitable due to allergies. If a person has a specific allergy, and a meal contains an ingredient that is excluded for that allergen, the rule infers that the meal is unsafe for that person.

2. Dietary Exclusion Rule

If a person follows a certain diet and a meal includes an ingredient that violates that diet, the system infers that the meal is not suitable.

3. Time Suitability Rule

Unlike the exclusion rules, this rule infers *suitability*. It matches meals to people based on the time they request. If a meal is served at a specific time and a person has requested that time, the system concludes that the meal is suitable for them.



5. Agile and Ontology-Based Meta-modelling

5.1 AOAME

This step was all about making sure the workflow diagram (BPMN) talks directly to our smart knowledge base (the Ontology in Protégé). I created a class **PreferenceProcessingTask**, which represents the BPMN task at the ontology level. I also added an instance **DetermineMealSuggestion_Instance.** This instance represents the moment in the restaurant's process where the system has to decide what meal to show the guest.

Task Input and Outputs:

- **Input (`handlesProfile`):** The guest's specific needs.
- **Output (`producesMeal`):** The recommended dish.

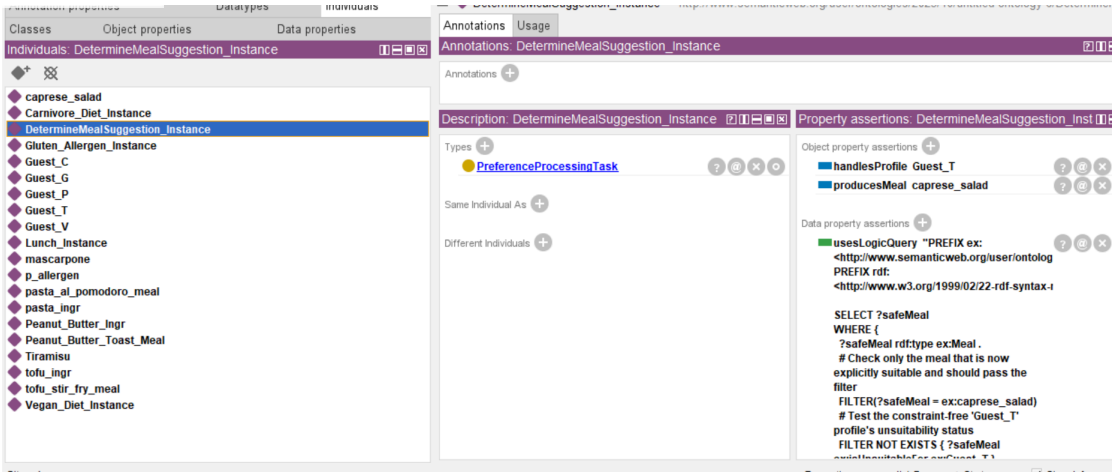
As for the execution logic a SPARQL query, which performs the meal selection, was directly stored in the ontology.

```
FILTER NOT EXISTS { ?safeMeal ex:isUnsuitableFor ex:Guest_T }
```

The query basically asks to show all the meals except the ones that are unsuitable for the guest. This filter allows the query to rely on Jena's reasoning results.

By combining the AOAME mapping with this filter, the BPMN task runs a query that automatically excludes all "unsafe" meals, providing a flexible personalized menu suggestion directly from the rules in Protégé.

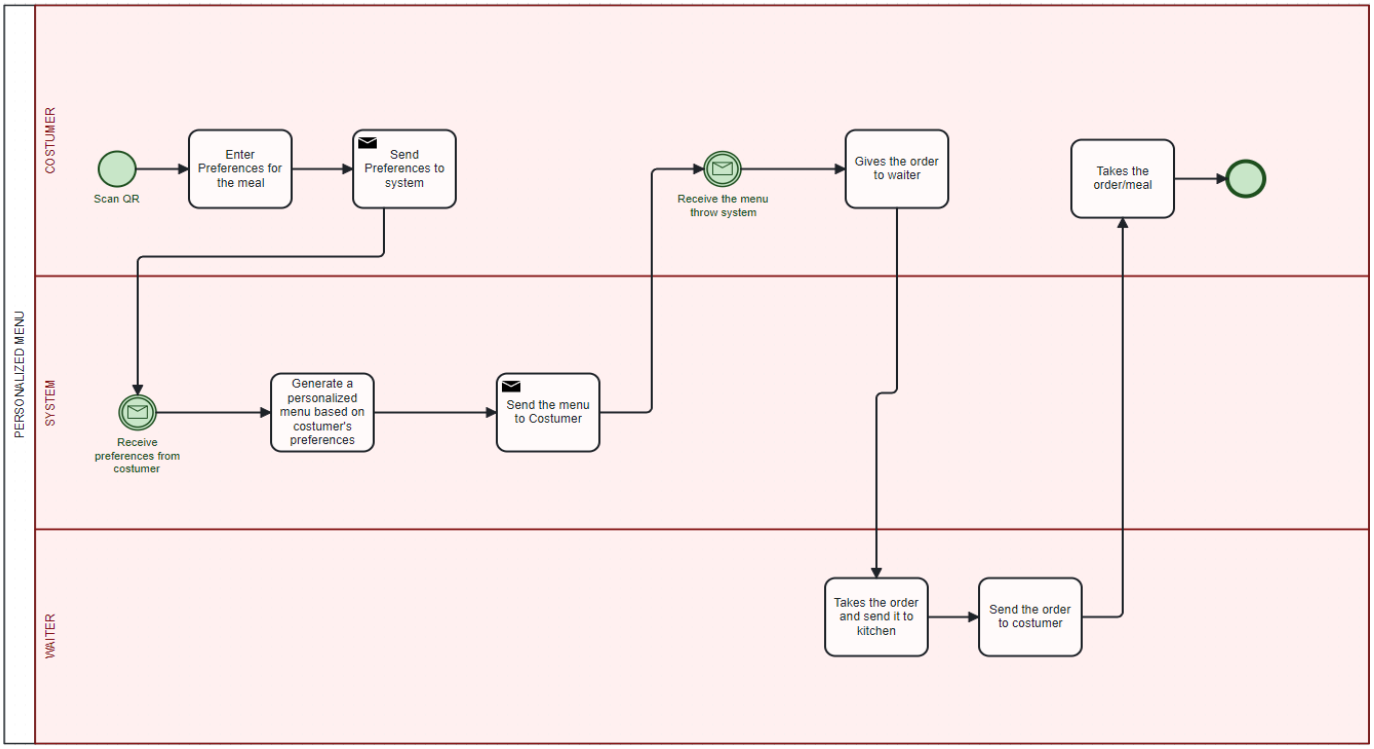
This setup creates a fully semantic and agile decision mechanism, where the business process task is executed by rules embedded in the knowledge base, giving flexibility to change rules without altering the process flow.



5.2 BPMN

I designed a BPMN process model to show how the personalized menu system will work, by implementing the personalized menu in a real restaurant scenario. It can be a way how a restaurant could optimize this process by automatizing it. The whole point of my BPMN was to eliminate the step where customers have to tell the waiter all their preferences one by one, and then wait for the waiter to suggest suitable meals. By implementing the personalized menu system, the "suggester" role is now handled automatically - customers enter their preferences directly, and the system instantly generates a customized menu for them.

The process diagram maps out the complete workflow from when a customer scans the QR code until they receive their food.



The process is divided into three swim lanes representing different actors: the customer, the system (which includes the knowledge base), and the waiter. It starts with the customer scanning a QR code and entering their preferences - things like dietary restrictions, allergies, or preferred meal times.

These preferences get sent to the personalized menu system, which uses the ontology and SPARQL queries we built earlier to generate a customized menu showing only suitable meals. The system automatically filters out anything the customer can't or doesn't want to eat. For example, if someone is vegan, they won't see any meals with dairy or meat. If they have a peanut allergy, dishes with peanuts are hidden. The filtered menu gets sent back to the customer's phone, they choose what they want, and the order goes to both the waiter and the kitchen.

This BPMN diagram essentially connects all the technical components we built (ontology, rules, queries) to an actual business process. It shows how the knowledge engineering work translates into a practical solution that improves the customer experience by giving them a personalized, easy-to-navigate menu instead of overwhelming them with options they can't choose anyway.

5.3 Jena Fuseki

The **DetermineMealSuggestion** task from the BPMN model has been successfully mapped to the AOAME semantic model, demonstrating proper integration of the business process into the semantic architecture. The task is linked to the input profile (**Guest_T**), which defines the type of user information it processes, the output meal (**caprese_salad**), which represents the result of the semantic execution, and the associated execution logic, which encodes the filtering or decision rules the agent uses. These confirmed links show that the task is structurally prepared to handle input, produce the correct output, and execute its reasoning as intended. All connections are correctly defined in Protégé and stored in Fuseki, providing complete structural proof of semantic task integration. The final step remaining is to configure the Java environment to run the Jena Reasoner, enabling the task to perform functional execution and dynamic meal filtering.


```
{
  "head": {
    "vars": [
      "profileName",
      "mealName",
      "querySnippet"
    ]
  },
  "results": {
    "bindings": [
      {
        "profileName": {
          "type": "literal",
          "value": "/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/Guest_T"
        },
        "mealName": {
          "type": "literal",
          "value": "/www.semanticweb.org/user/ontologies/2025/10/untitled-ontology-6/caprese_salad"
        },
        "querySnippet": {
          "type": "literal",
          "value": "PREFIX ex: <http://www.semanticweb.org/user/ontolo"
        }
      }
    ]
  }
}
```

In this project component, I utilized GraphDB as the core platform for managing and querying our structured data. To host and access the SPARQL endpoints, I utilized Apache Jena Fuseki as the triplestore server, which seamlessly integrated with my development environment.

The implementation focused on developing various SPARQL (SPARQL Protocol and RDF Query Language) queries. These queries were essential for data retrieval, graph traversal, and relationship extraction within the ontology. This work validated the efficiency of using an RDF knowledge graph approach to access and manipulate complex, interconnected data structures for subsequent project stages.

6. Conclusion

6.1 Conclusion (Angjelina Bahushi)

The personalized menu system project combines multiple knowledge engineering approaches to solve a real problem in the restaurant industry. By using **Prolog** for logical rules, **SWRL** for inferencing, an **ontology hosted on a triplestore like Jena Fuseki**, **SPARQL queries** for retrieval, **SHACL validation** for data quality, and **BPMN** process modeling, we built a system that can recommend meals based on individual customer needs while ensuring food safety and efficiency.

Advantages of Knowledge-Based Solutions

1. **Personalization:** The system provides highly tailored menu recommendations by filtering meals according to each customer's allergies, specific dietary preferences, and meal time requirements, making the dining experience more convenient.
2. **Safety: SHACL validation** and ontology rules, particularly those implemented in **Protege**, ensure that customers with allergies never see meals containing their specific allergens, which is critical for health and safety.
3. **Transparency**: The knowledge graph, queried via **SPARQL**, makes it easy to understand why a meal was recommended or excluded—you can trace back through ingredients, dietary types, and restrictions to see the exact reasoning.
4. **Flexibility:** Adding new meals, ingredients, or complex dietary restrictions is straightforward because the ontology-based approach keeps everything organized and connected logically.
5. **Business Process Integration:** The **BPMN model** clearly shows how the technical solution fits into real restaurant operations, making it easier to implement and understand for non-technical staff. Also by implementing BPMN we all can see how everything gets easier even for the restaurant's staff but also for the clients. Everything that before was done manually, now it's automated.

Disadvantages of Knowledge-Based Solutions

1. **Technical Complexity:** Building ontologies, writing complex **SPARQL** queries, and creating **SHACL** shapes requires specialized knowledge that not everyone has, making development and maintenance challenging.
2. **Initial Setup Time:** Getting the ontology structure right and testing all the rules (in Prolog, SWRL, and SHACL) takes considerable time upfront compared to simpler database approaches.
3. **Performance Concerns:** As the number of meals and guests increases, reasoning and query execution within the triplestore (**GraphDB/Jena Fuseki**) might slow down without proper optimization and indexing.
4. **Tool Dependencies:** The system relies on specific, advanced tools like Protege for ontology editing and **GraphDB/Jena Fuseki** for triple storage, which means you're locked into these specific technologies.
5. **Data Quality Requirements:** The whole system depends on having accurate, complete data—if someone forgets to mark an ingredient as containing an allergen, the system could make dangerous recommendations.

Final Thoughts

Overall, this project showed me that knowledge-based systems are incredibly powerful for domains where relationships and rules matter more than simple data storage. The ability to ask complex questions like "which meals are safe for a vegan guest who wants lunch and has a gluten allergy" and get accurate answers automatically is really valuable.

However, I also learned that these systems require careful planning and ongoing maintenance. You can't just set them up and forget about them—the ontology needs updates when new ingredients or dietary types appear, queries might need optimization as data grows, and validation rules need testing whenever changes are made.

For a restaurant wanting to implement this, the benefits of better customer experience and food safety probably outweigh the technical challenges, especially if they already have IT staff familiar with Semantic Web technologies. The important thing is balancing a smart solution with practical reality - considering how much it costs to maintain and whether staff can actually be trained to use it properly.

7. Conclusion (Elva Panariti)

The main goal of this project was to build an intelligent menu system that uses smart rules to recommend meals based on a guest`s needs. The most exciting and difficult part at the same time was bringing together several technologies and tools to build this smart system.

7.1 Advantages and Disadvantages

Using multiple tools gave me powerful results, but it also made things complicated. Furthermore, I am going to describe the good news and the hard truths regarding the tools used in this project.

7.1.2 Decision Tables (DMN)

Decision Model and Notation (DMN) provided the rule-based logic for filtering meals based on guest preferences and dietary restrictions. This standardized approach to decision-making became the first layer of my recommendation engine.

Pros:

- Everyone can read and understand decision tables, even people who don't code
- The Collect feature made it easy to gather all matching meals in one go
- Changes to rules don't require changing code, just update the table

Cons:

- There's no built-in testing environment, so I had to manually check if my rules worked correctly
- Testing with real-world scenarios was difficult and time-consuming

7.1.3 Knowledge Database (Protégé + SHACL + OWL)

The semantic layer—built using Protégé for ontology design, OWL for defining relationships, and SHACL for data validation formed the intelligent core of the system. This knowledge base enabled the system to understand not just data, but the meaning and relationships between meals, ingredients, and dietary requirements.

Pros:

- The visual diagrams helped me see how everything connected
- The system can reason about relationships I did not explicitly program
- Once set up correctly, it prevents bad data from getting in

Cons:

- Learning to think in terms of ontologies was really difficult at first
- I had to manually type in all the meals and ingredients, which took forever
- Small mistakes in data entry created big problems later

7.1.4 Connecting Everything (AOAME + BPMN + Fuseki)

BPMN orchestrated the workflow, defining how different components interact throughout the recommendation process. AOAME provided the framework for integrating business processes with semantic reasoning, while Apache Jena Fuseki served as the SPARQL endpoint for querying the knowledge base. Together, these tools created a bridge between process logic and semantic knowledge.

Pros:

- The workflow diagrams made it clear how the system works
- I could show the process to anyone and they would understand it
- The connection between workflows and the knowledge base let me build dynamic, smart queries
- Everything was traceable. I could see exactly what happened at each step

Cons:

- Getting all the pieces to talk to each other was much harder than I expected
- Performance was slow when dealing with large amounts of data

7.1.5 The complete system

Pros:

- The combination of DMN and semantic reasoning gave me precise filtering
- For health-critical things like allergies, the system is highly reliable
- I successfully showed that these technologies can work together

- The system handles complex requirements that would be nearly impossible to code manually

Cons:

- The system is complex
- You need to understand DMN, ontologies, BPMN, SPARQL, and Prolog to maintain it
- If something breaks, figuring out where the problem is can be really difficult

7.2 Future Improvements

While this project successfully demonstrates the potential of knowledge engineering technologies, there are several areas where I would do things differently:

Expanding the Knowledge Base: The ontology currently contains a limited set of meal and ingredient individuals. I would add more rules and complex relationship so the system could me more sophisticated.

Automation of Data Entry: One of the most time-consuming aspects was manually entering individuals into Protégé. Future work should focus on developing automated import tools that can populate the ontology from external sources.

User Interface Development: Currently, the system operates primarily as a backend service. Creating an intuitive front-end interface where restaurant staff or guests can interact with the recommendation system would make it practical for real-world use.

7.3 Final Conclusion

This project taught me that combining multiple knowledge engineering tools can create something really powerful. The system we built can handle complex dietary requirements and make smart recommendations that would be nearly impossible to program manually.

However, I also learned that more tools doesn't always mean better. The complexity made development slower and would make long-term maintenance challenging.

Despite the challenges, I'm proud of what we built. The system works, it's precise, and it demonstrates that semantic technologies can solve real-world problems. With some

simplification and automation, this could absolutely be used in a real restaurant or catering business.

8. Conclusion from both of us

This project was way more challenging than we expected, but we learned so much by actually doing it rather than just hearing about it in lectures.Our professor's lectures gave us the basic concepts ,what ontologies are, how semantic reasoning works, why knowledge graphs matter, but there is a huge difference between understanding something in theory and actually making it work.

The best part about working as a team was that when one person got stuck, the other could help figure it out. We divided up the work ,one person focused more on the ontology while the other worked on queries and validation but we had to constantly communicate to make sure everything would connect properly. Sometimes one of us would get something working and explain it to the other, and teaching each other actually helped us understand things better.

What We Actually Learned

Beyond the technical stuff, we learned how to break down a complicated problem, research things we'd never seen before, and keep trying when things didn't work the first time. We faced with new tools and technology and by communicating with each other everything got easier and more interesting . We had a basic knowledge from other classes ,like BPMN, and we found it very interesting implementing our knowledge and skills in our project.

We are proud of what we built, not because it's perfect, but because we actually got it working despite all the challenges. The project showed us that knowledge engineering can solve real problems, but it also showed us that it takes patience, teamwork, and a lot of trial and error. Our professor gave us the foundation, our research filled in the gaps, and working together made it actually happen.

This documentation was written by both of us, based on the work we have done on this project. Below is the GitHub link : https://github.com/elvapanariti/Kebi_Project