University of Camerino

SCHOOL OF SCIENCE AND TECHNOLOGY

Master degree in Computer Science (LM-18)
Knowledge Engineering and Business Intelligence

Intelligent Meal Suggestion System for Restaurants Using Knowledge-Based Solutions and Ontology-Driven Meta-Modelling

Authors:
Chiocchi Francesco
Rossini Nicolò

Supervisors:
Prof. Knut Hinkelmann
Prof. Emanuele Laurenzi

A.Y. 2023/2024

Contents

Listings

List of Figures

1. Introduction

This document aims to outline the process undertaken to complete the Knowledge Engineering and Business Intelligence project. Section 1.1 will provide a description of the system to be developed, while Section 1.2 will define the tasks to be accomplished. Following this, we will examine the System Design phase (Chapter 2), the implemented Knowledge-Based Solutions (Chapter 3), and the description of our implementation of Agile and Ontology-based Meta-Modelling.
In the Conclusion chapter (Chapter 5), we will offer our personal perspectives on the solutions presented.

The solution developed for Task 1 can be found at this link:
https://github.com/FrancescoChiocchi8/KEBI-Project, while the solution for Task 2 can be found at this link: https://github.com/FrancescoChiocchi8/Ontology4Modeling Environment

1.1 Project Description

Many restaurants have their menus digitized. Guests can scan a QR code and have the menu presented on their smartphones. A disadvantage is that the screen is very small and it is difficult to get an overview, in particular, if the menu is large. However, some guests can not or do not want every meal, e.g. vegetarians or guests with an allergy. Instead of showing all the meals that are offered, it would be preferable to show only those meals the guest prefers. The objective of the project is to represent the knowledge about meals and guest preferences and create a system that allows to select those meals that fit the guest preferences. The knowledge base shall contain information about typical meals of an Italian restaurant, e.g. pizza, pasta, and main dishes. Meals consist of ingredients. There are different types of ingredients like meat, vegetables, fruits, or dairy. For each ingredient, there is information about the calories. Guests can be carnivores, vegetarians, calorie-conscious, or suffer from allergies, e.g. lactose or gluten intolerance.

1.2 Project Tasks

The following are the tasks to tackle for completing the Knowledge Engineering and Business Intelligence projects:

1. Create different knowledge-based solutions for recommending food depending on the profile of a guest (carnivores, vegetarians, calorie-conscious, suffering from allergiesetc.) usind the following representation languages:

- Decision tables (including DRD with sub-decisions and corresponding decision tables);
- Prolog (including facts and rules);
- Knowledge graph/Ontology (including rules in SWRL, queries in SPARQL and SHACL shapes)

2. Agile and ontology-based meta-modelling: adapt BPMN 2.0 to suggest the meals for a given customer. For this, you can re-use or extend the knowledge graph/ontology created in the previous task. One option that you have is to specify the class BPMN Task with a new class and add additional properties, similar to what we have done in class with the Business Process as a Service case. Think of a new graphical notation for the new modelling element, which could be easy to understand for the restaurant manager. Use the triple store interface (Jena Fuseki) to fire the query result.

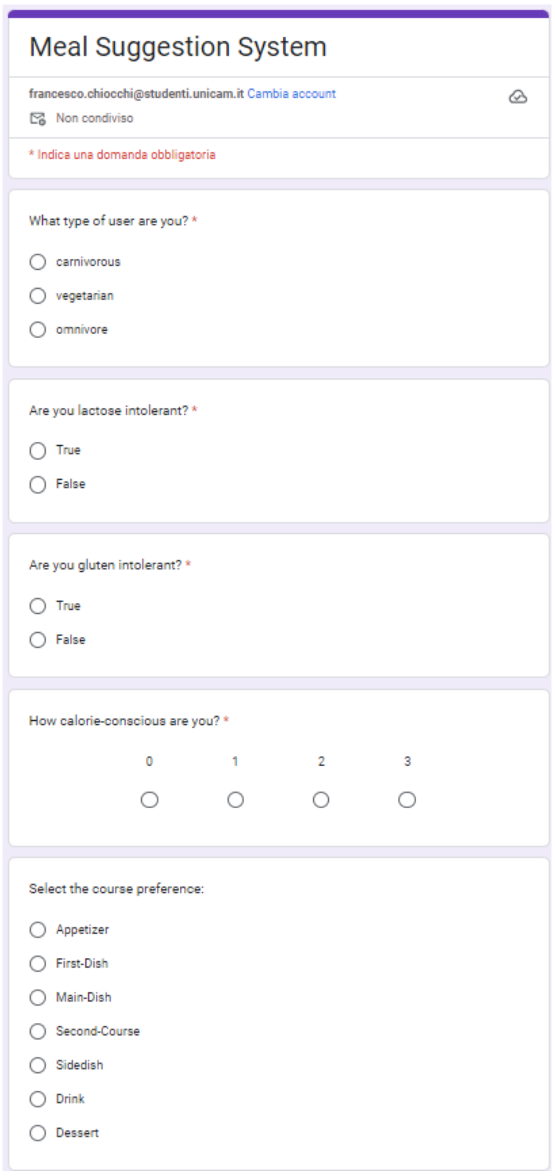
2. System Design

In this chapter, we will present our concepts regarding the data inputs and outputs that the system will analyze.

2.1 Input format

We have designed an input form, thanks to Google Modules and Camunda (for the output view), that will serve as the foundation for the rest of this project. An image illustrating our concept of the input form is shown below (Figure 2.1). This form is intended to be completed by the customer to generate a list of the best meals based on their preferences.

In this form, presented in Figure 2.1, the first choice the user must make is to select either 'carnivorous,' 'vegetarian,' or 'omnivore' (represented by circles, indicating that only one option can be selected). This selection is mandatory.
The user must also indicate any intolerances (lactose or gluten).
Further down, the user selects their interest level in consuming high-calorie foods (level 0), medium-high calorie foods (level 1), medium-low calorie foods (level 2), or low-calorie foods (level 3). This selection is also mandatory, with level 0 set as the default.
Finally, the user can complete the corresponding section by selecting the course type, such as 'Appetizer', 'First Course', 'Main Course', 'Pizza', 'Drink', or 'Dessert'.



Meal Suggestion System

francesco.chiocchi@studenti.unicam.it Cambia account

Non condiviso

* Indica una domanda obbligatoria

What type of user are you? *

◯ carnivorous
◯ vegetarian
◯ omnivore

Are you lactose intolerant? *

◯ True
◯ False

Are you gluten intolerant? *

◯ True
◯ False

How calorie-conscious are you? *

0	1	2	3
◯	◯	◯	◯

Select the course preference:

◯ Appetizer
◯ First-Dish
◯ Main-Dish
◯ Second-Course
◯ Sidedish
◯ Drink
◯ Dessert

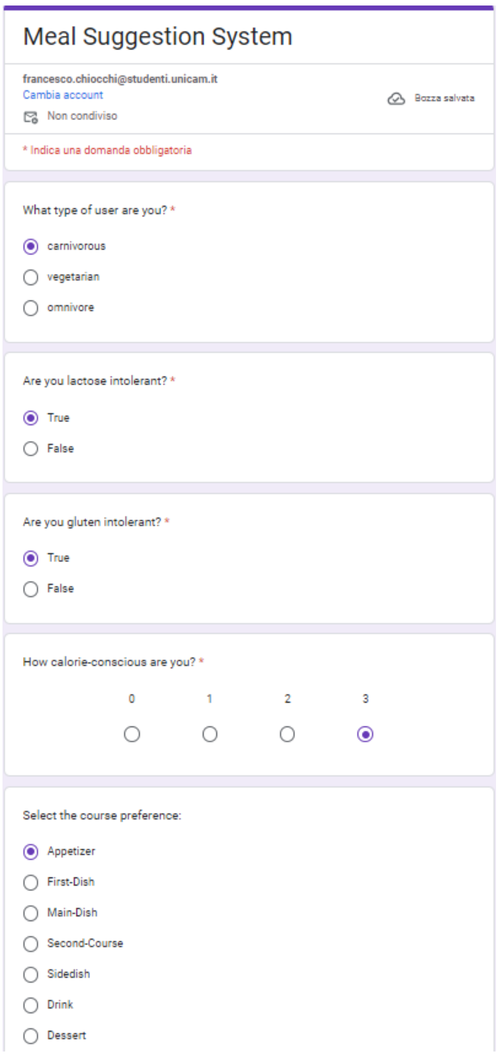
Figure 2.1: Sample Input form for the customer

Suggested Meals
Affettati Misti

Table 2.1: Output Preference for the Figure 2.2

2.2 Output format

In the Figure 2.2 we can see an example of how the form might be completed by the user.



Meal Suggestion System

francesco.chiocchi@studenti.unicam.it
Cambia account
Non condiviso
Bozza salvata

* Indica una domanda obbligatoria

What type of user are you? *

[✓ carnivorous]
vegetarian
omnivore

Are you lactose intolerant? *

[✓ True]
False

Are you gluten intolerant? *

[✓ True]
False

How calorie-conscious are you? *

0 1 2 [✓ 3]

Select the course preference:

[✓ Appetizer]
First-Dish
Main-Dish
Second-Course
Sidedish
Drink
Dessert

Figure 2.2: Sample filled out form

3. Knowledge Based Solution

Relying on the input form and format previously defined, we can query our knowledge base in three different ways. Firstly, by using Decision Tables (Section 3.1) to match user preferences and filter out undesired meal. With Prolog (Section 3.2), we can extend the logic provided by the Decision Tables by applying the functionalities provided by a programming language. This allows the users to manipulate data and perform even more actions. Finally, using Knowledge Graphs (Section 3.3, we can store our data in a convenient format and query them as if they were stored in a database. Following, in this chapter, we will discuss each of the three solutions, their main advantages and drawbacks.

3.1 Decision Model

In the realm of business analysis, the Decision Model and Notation (DMN), as outlined by the Object Management Group (OMG), stands as a recognized standard. This methodology finds application in delineating and structuring recurring decisions within organizations, facilitating a format where decision models can be seamlessly shared across entities. By adhering to this standard, companies can establish a unified modeling notation, empowering them to make decisions conducive to effective decision management and adherence to business rules.

DMN Elements

The DMN standard comprises four key elements:

- Decision Requirements Diagrams: These diagrams elucidate the interconnections among decision-making components, forming a dependency network.
- Decision Tables[1]: These tables offer a succinct visual depiction for specifying actions contingent upon provided conditions.
- Business Context: This pertains to the contextual factors influencing the decisions within the company.
- Friendly Enough Expression Language (FEEL): FEEL serves as a language utilized for evaluating expressions within decision tables.

3.1.1 Camunda

Camunda [2] is an open-source platform, built in Java, specifically engineered to facilitate workflow and process automation, catering to companies of varying scales. Its func-

[1]Decision Table: https://en.wikipedia.org/wiki/Decision_table

[2]Camunda:https://camunda.com/

tionalities extend to the creation of decision models and, more broadly, BPMN process diagrams and DMN decision tables. Additionally, Camunda offers[3] REST APIs, enabling developers who do not utilize Java to access its features. We leveraged Camunda to establish our knowledge base regarding meals and to delineate decision-making processes and rules congruent with our organizational goals.

3.1.2 Our Decision Model

As illustrated in Figure 3.1, our model comprises two Decision Tables and one Literal Expression. Unlike in the first submission, we first applied restrictions to the ingredients, followed by restrictions on the meals. The following section provides a detailed explanation of each table.

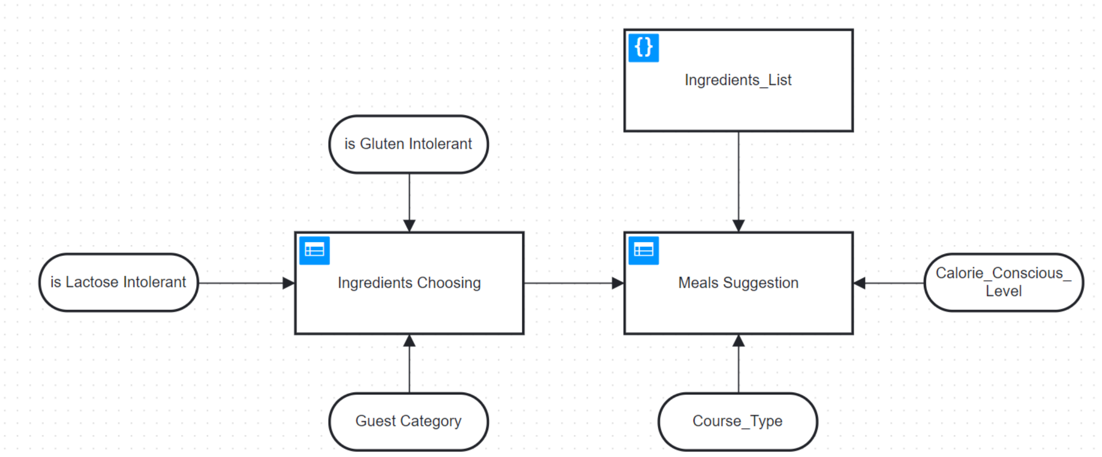


Figure 3.1: Our Decision Model in Camunda

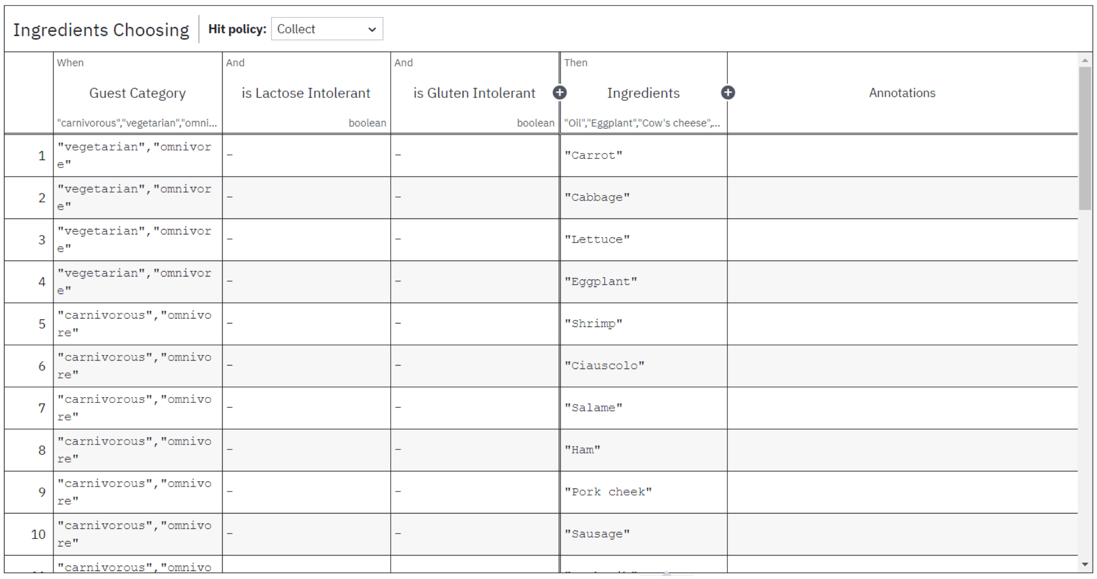
3.1.3 Decision Tables

Now, referring to Figure 3.1, let's describe all the Decision Tables and their respective inputs and outputs:

- *Ingredient Choosing*: This takes several input parameters, including the user's dietary category—whether they are omnivorous, carnivorous, or vegetarian. It also considers a boolean parameter (true or false) indicating whether the user is lactose intolerant, as well as another boolean parameter for gluten intolerance. This decision table then returns a list of meals filtered based on the user's preferences. As illustrated in Figures 3.2 and 3.3, the input for Guest Category includes three different parameters: Carnivorous, for users who follow a meat-based diet without consuming vegetables; Vegetarian, for users who avoid meat; and Omnivore, for users who consume both meat and vegetables. We have designed this structure to enhance the table's extensibility. For instance, if in the future there is a need to include a Vegan category, it can easily be added alongside the existing categories, allowing for the identification of vegan ingredients. Regarding the boolean inputs, if a hyphen ("-") is present in the Decision Table, it indicates that the specific ingredient does not contain allergens. Conversely, if 'false' is

[3] Camunda Documentation https://docs.camunda.org/manual/7.19/reference/dmn/feel/

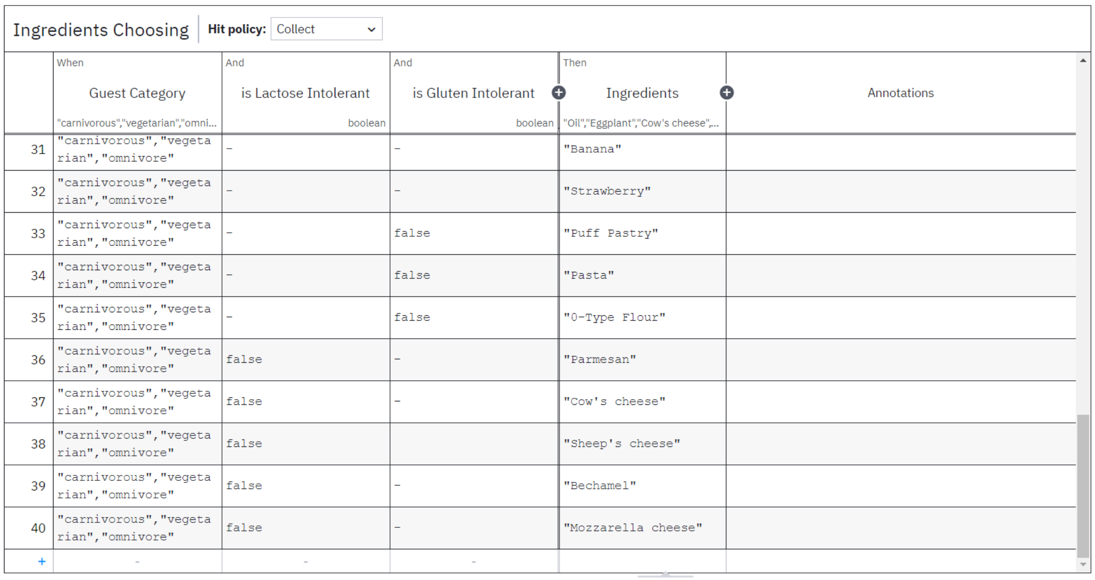
entered, it signifies that the ingredient does contain an allergen. In such cases, if the user has that specific intolerance, the ingredient will be excluded, and meals containing the allergen will not be suggested to the user.



Ingredients Choosing | Hit policy: Collect

	When	And	And	Then	
	Guest Category	is Lactose Intolerant	is Gluten Intolerant	Ingredients	Annotations
	"carnivorous","vegetarian","omni...	boolean	boolean	"Oil","Eggplant","Cow's cheese",...	
1	"vegetarian","omnivore"	-	-	"Carrot"	
2	"vegetarian","omnivore"	-	-	"Cabbage"	
3	"vegetarian","omnivore"	-	-	"Lettuce"	
4	"vegetarian","omnivore"	-	-	"Eggplant"	
5	"carnivorous","omnivore"	-	-	"Shrimp"	
6	"carnivorous","omnivore"	-	-	"Ciauscolo"	
7	"carnivorous","omnivore"	-	-	"Salame"	
8	"carnivorous","omnivore"	-	-	"Ham"	
9	"carnivorous","omnivore"	-	-	"Pork cheek"	
10	"carnivorous","omnivore"	-	-	"Sausage"	
	"carnivorous","omnivo				

Figure 3.2: Ingredients Choosing -1- DT

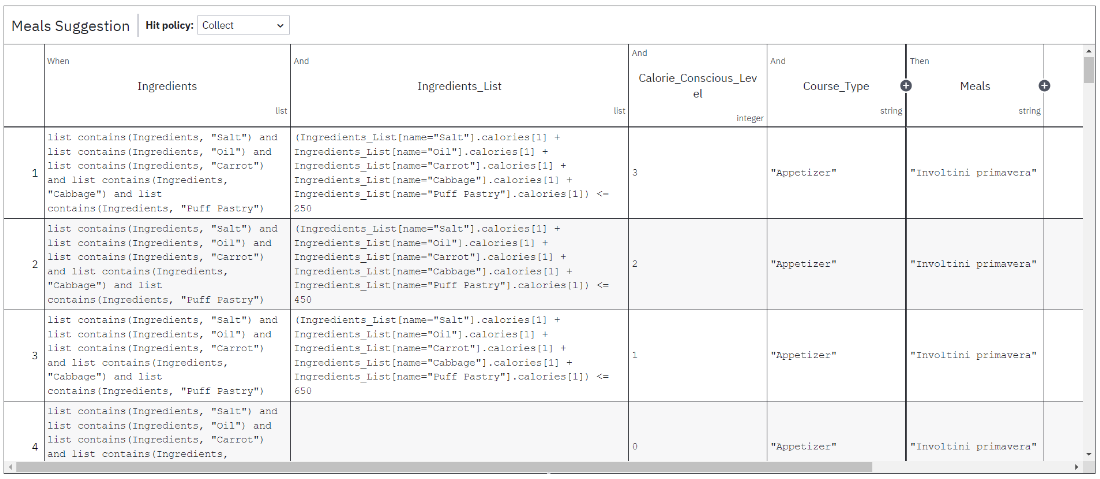


Ingredients Choosing | Hit policy: Collect

	When	And	And	Then	
	Guest Category	is Lactose Intolerant	is Gluten Intolerant	Ingredients	Annotations
	"carnivorous","vegetarian","omni...	boolean	boolean	"Oil","Eggplant","Cow's cheese",...	
31	"carnivorous","vegetarian","omnivore"	-	-	"Banana"	
32	"carnivorous","vegetarian","omnivore"	-	-	"Strawberry"	
33	"carnivorous","vegetarian","omnivore"	-	false	"Puff Pastry"	
34	"carnivorous","vegetarian","omnivore"	-	false	"Pasta"	
35	"carnivorous","vegetarian","omnivore"	-	false	"0-Type Flour"	
36	"carnivorous","vegetarian","omnivore"	false	-	"Parmesan"	
37	"carnivorous","vegetarian","omnivore"	false	-	"Cow's cheese"	
38	"carnivorous","vegetarian","omnivore"	false		"Sheep's cheese"	
39	"carnivorous","vegetarian","omnivore"	false	-	"Bechamel"	
40	"carnivorous","vegetarian","omnivore"	false	-	"Mozzarella cheese"	
+	-	-	-		

Figure 3.3: Ingredients Choosing -2- DT

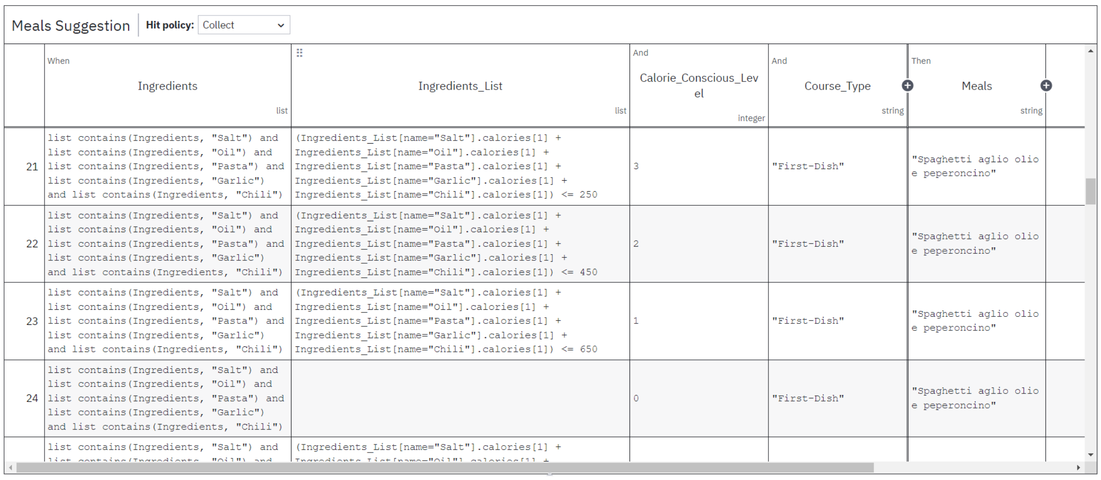
- *Meals Suggestion*: This table takes two inputs from the user: one derived from the previous table and another from the Literal Expression. The first input, Course_Type, refers to the type of meal course (e.g., Appetizer, First-Dish, etc.), and the second, Calorie_Conscious_Level, represents the user's calorie-conscious level, ranging from 0 to 3. The table returns a list of meals filtered according to this level, with a default value of 0. A level 0 indicates no concern about calorie content, while levels 1, 2, and 3 indicate increasing concern, with level 3 corresponding to a preference for meals under 250 kcal.



Meals Suggestion | **Hit policy:** Collect

	When Ingredients list	And Ingredients_List list	And Calorie_Conscious_Level integer	And Course_Type string	Then Meals string
1	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Carrot") and list contains(Ingredients, "Cabbage") and list contains(Ingredients, "Puff Pastry")	(Ingredients_List[name="Salt"].calories[1] + Ingredients_List[name="Oil"].calories[1] + Ingredients_List[name="Carrot"].calories[1] + Ingredients_List[name="Cabbage"].calories[1] + Ingredients_List[name="Puff Pastry"].calories[1]) <= 250	3	"Appetizer"	"Involtini primavera"
2	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Carrot") and list contains(Ingredients, "Cabbage") and list contains(Ingredients, "Puff Pastry")	(Ingredients_List[name="Salt"].calories[1] + Ingredients_List[name="Oil"].calories[1] + Ingredients_List[name="Carrot"].calories[1] + Ingredients_List[name="Cabbage"].calories[1] + Ingredients_List[name="Puff Pastry"].calories[1]) <= 450	2	"Appetizer"	"Involtini primavera"
3	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Carrot") and list contains(Ingredients, "Cabbage") and list contains(Ingredients, "Puff Pastry")	(Ingredients_List[name="Salt"].calories[1] + Ingredients_List[name="Oil"].calories[1] + Ingredients_List[name="Carrot"].calories[1] + Ingredients_List[name="Cabbage"].calories[1] + Ingredients_List[name="Puff Pastry"].calories[1]) <= 650	1	"Appetizer"	"Involtini primavera"
4	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Carrot") and list contains(Ingredients,		0	"Appetizer"	"Involtini primavera"

Figure 3.4: Meals Suggestion -2- DT



Meals Suggestion | **Hit policy:** Collect

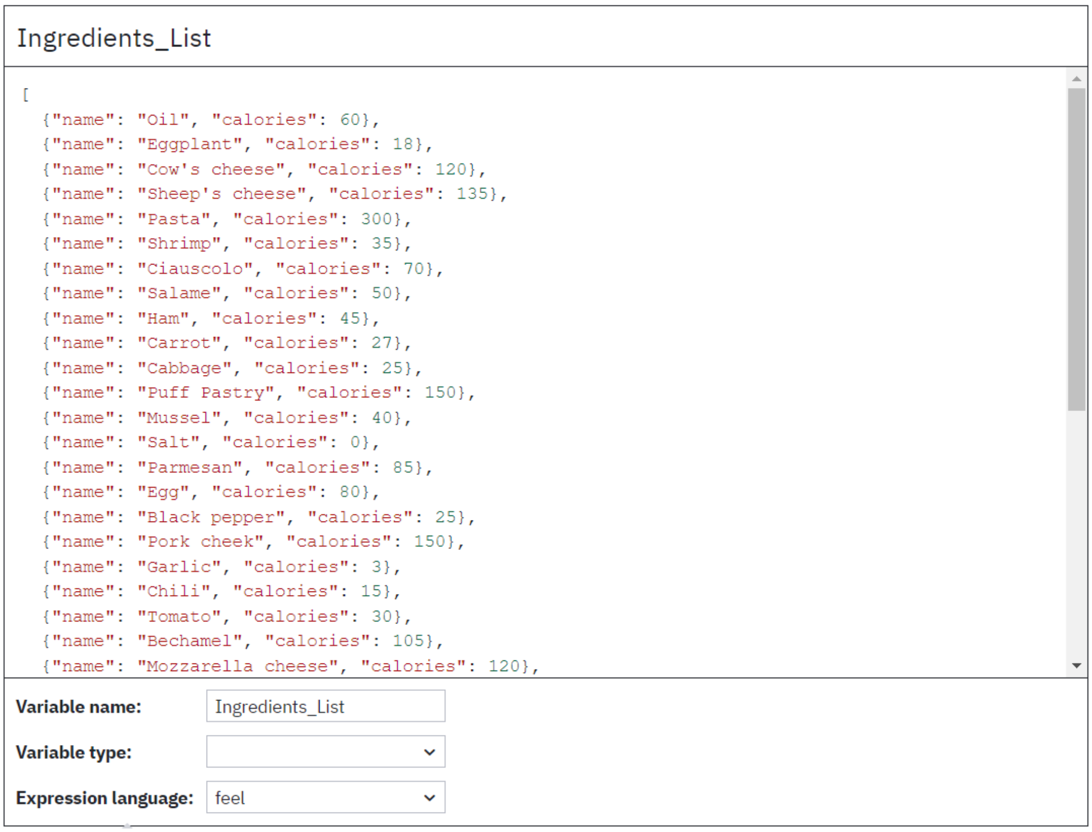
	When Ingredients list	Ingredients_List list	And Calorie_Conscious_Level integer	And Course_Type string	Then Meals string
21	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Pasta") and list contains(Ingredients, "Garlic") and list contains(Ingredients, "Chili")	(Ingredients_List[name="Salt"].calories[1] + Ingredients_List[name="Oil"].calories[1] + Ingredients_List[name="Pasta"].calories[1] + Ingredients_List[name="Garlic"].calories[1] + Ingredients_List[name="Chili"].calories[1]) <= 250	3	"First-Dish"	"Spaghetti aglio olio e peperoncino"
22	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Pasta") and list contains(Ingredients, "Garlic") and list contains(Ingredients, "Chili")	(Ingredients_List[name="Salt"].calories[1] + Ingredients_List[name="Oil"].calories[1] + Ingredients_List[name="Pasta"].calories[1] + Ingredients_List[name="Garlic"].calories[1] + Ingredients_List[name="Chili"].calories[1]) <= 450	2	"First-Dish"	"Spaghetti aglio olio e peperoncino"
23	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Pasta") and list contains(Ingredients, "Garlic") and list contains(Ingredients, "Chili")	(Ingredients_List[name="Salt"].calories[1] + Ingredients_List[name="Oil"].calories[1] + Ingredients_List[name="Pasta"].calories[1] + Ingredients_List[name="Garlic"].calories[1] + Ingredients_List[name="Chili"].calories[1]) <= 650	1	"First-Dish"	"Spaghetti aglio olio e peperoncino"
24	list contains(Ingredients, "Salt") and list contains(Ingredients, "Oil") and list contains(Ingredients, "Pasta") and list contains(Ingredients, "Garlic") and list contains(Ingredients, "Chili")		0	"First-Dish"	"Spaghetti aglio olio e peperoncino"
	list contains(Ingredients, "Salt") and	(Ingredients_List[name="Salt"].calories[1] +			

Figure 3.5: Meals Suggestion -1- DT

As shown in Figures 3.4 and 3.5, the adopted Hit policy is Collect, which aggregates results based on certain criteria. The table applies four checks for each meal, corresponding to the user's calorie-conscious level. While fewer levels could have been used, four levels offer greater accuracy and restriction. The first column, Ingredients, verifies whether all required ingredients are available; for example, if a user is gluten intolerant, Puff Pastry will be excluded, and the meal will not be suggested. Similarly, lactose intolerance is considered. Moving right in the figure, the total calories of the meal are compared against a threshold that varies by level: less than 250 kcal for level 3, less than 450 kcal for level 2, and less than 650 kcal for level 1. No calorie check is performed for level 0. In addition to the calorie-conscious level, the user also inputs the Course_Type, which helps further filter the meals to provide a list of ideal options. The output of the Decision Table is a list of meals tailored to the user's preferences.

3.1.4 Literal Expression

To streamline our process, we have developed a Literal Expression that encapsulates the ingredients and their corresponding calorie values in the form of a Map, where each ingredient name is associated with a numerical value representing its calorie content. Below, a portion of the Literal Expression 3.6 is shown.



Ingredients_List

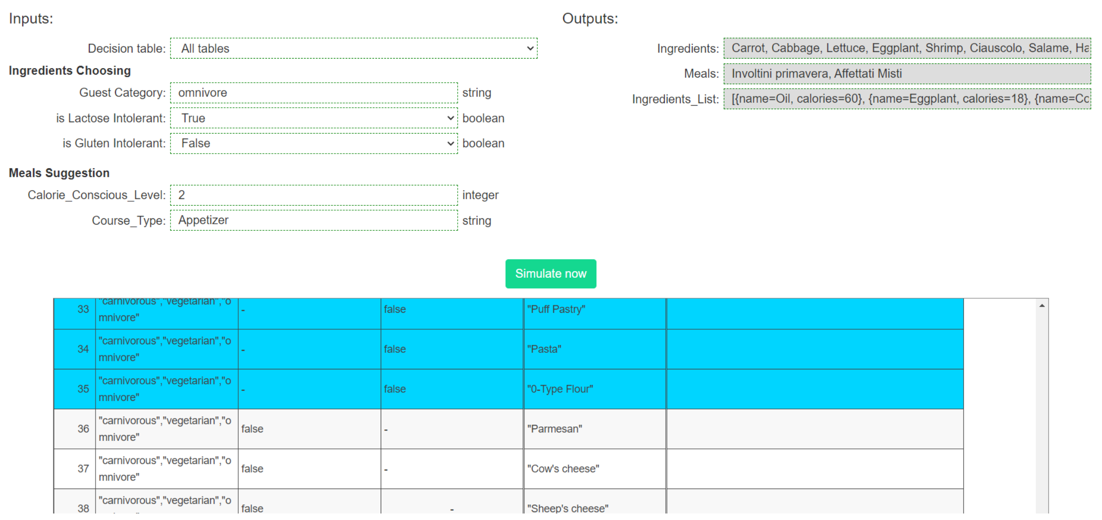
```
[
  {"name": "Oil", "calories": 60},
  {"name": "Eggplant", "calories": 18},
  {"name": "Cow's cheese", "calories": 120},
  {"name": "Sheep's cheese", "calories": 135},
  {"name": "Pasta", "calories": 300},
  {"name": "Shrimp", "calories": 35},
  {"name": "Ciauscolo", "calories": 70},
  {"name": "Salame", "calories": 50},
  {"name": "Ham", "calories": 45},
  {"name": "Carrot", "calories": 27},
  {"name": "Cabbage", "calories": 25},
  {"name": "Puff Pastry", "calories": 150},
  {"name": "Mussel", "calories": 40},
  {"name": "Salt", "calories": 0},
  {"name": "Parmesan", "calories": 85},
  {"name": "Egg", "calories": 80},
  {"name": "Black pepper", "calories": 25},
  {"name": "Pork cheek", "calories": 150},
  {"name": "Garlic", "calories": 3},
  {"name": "Chili", "calories": 15},
  {"name": "Tomato", "calories": 30},
  {"name": "Bechamel", "calories": 105},
  {"name": "Mozzarella cheese", "calories": 120},
```

Variable name:	Ingredients_List
Variable type:	
Expression language:	feel

Figure 3.6: Ingredients List Literal Expression

3.1.5 Camunda Simulation

Finally, we present an example of meal suggestions from our model using the Camunda DMN Simulator [4]. For this first simulation, we set the guest category to omnivore, selected lactose intolerance, set the calorie-conscious level to 2, and chose Appetizer as the preferred course. As shown in Figure 3.7 the output returned is 'Involtini Primavera' and 'Affettati Misti'.



Inputs:

Decision table:	All tables	
Ingredients Choosing		
Guest Category:	omnivore	string
is Lactose Intolerant:	True	boolean
is Gluten Intolerant:	False	boolean
Meals Suggestion		
Calorie_Conscious_Level:	2	integer
Course_Type:	Appetizer	string

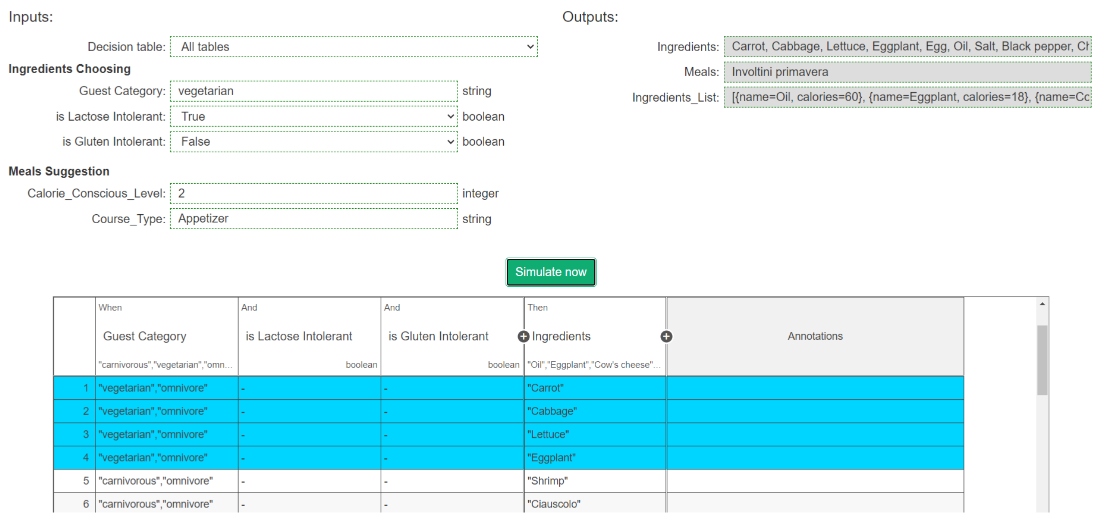
Outputs:

Ingredients:	Carrot, Cabbage, Lettuce, Eggplant, Shrimp, Ciauscolo, Salame, Ha
Meals:	Involtini primavera, Affettati Misti
Ingredients_List:	[{name=Oil, calories=60}, {name=Eggplant, calories=18}, {name=Cc

33	"carnivorous","vegetarian","omnivore"	-	false	"Puff Pastry"	
34	"carnivorous","vegetarian","omnivore"	-	false	"Pasta"	
35	"carnivorous","vegetarian","omnivore"	-	false	"0-Type Flour"	
36	"carnivorous","vegetarian","omnivore"	false	-	"Parmesan"	
37	"carnivorous","vegetarian","omnivore"	false	-	"Cow's cheese"	
38	"carnivorous","vegetarian","o	false	-	"Sheep's cheese"	

Figure 3.7: DMN First Simulation

If we now change the Guest Category to vegetarian, 'Affettati Misti' will no longer be displayed, leaving only 'Involtini Primavera', as it is lactose-free and contains only vegetarian ingredients, as shown in Figure 3.8.



Inputs:

Decision table:	All tables	
Ingredients Choosing		
Guest Category:	vegetarian	string
is Lactose Intolerant:	True	boolean
is Gluten Intolerant:	False	boolean
Meals Suggestion		
Calorie_Conscious_Level:	2	integer
Course_Type:	Appetizer	string

Outputs:

Ingredients:	Carrot, Cabbage, Lettuce, Eggplant, Egg, Oil, Salt, Black pepper, Cł
Meals:	Involtini primavera
Ingredients_List:	[{name=Oil, calories=60}, {name=Eggplant, calories=18}, {name=Cc

	When	And	And	Then	
	Guest Category	is Lactose Intolerant	is Gluten Intolerant	Ingredients	Annotations
	"carnivorous","vegetarian","omn…	boolean	boolean	"Oil","Eggplant","Cow's cheese"…	
1	"vegetarian","omnivore"	-	-	"Carrot"	
2	"vegetarian","omnivore"	-	-	"Cabbage"	
3	"vegetarian","omnivore"	-	-	"Lettuce"	
4	"vegetarian","omnivore"	-	-	"Eggplant"	
5	"carnivorous","omnivore"	-	-	"Shrimp"	
6	"carnivorous","omnivore"	-	-	"Ciauscolo"	

Figure 3.8: DMN Second Simulation

[4] DMN Simulator Reference Page: https://consulting.camunda.com/dmn-simulator/

3.2 Prolog

Logic programming utilizes logic as a declarative representation language and backward chaining as an inference rule. Prolog[5] is a logic programming language associated with artificial intelligence and computational linguistics. The strength of this language lies in its logical foundation. Prolog syntax provides:

- **Symbols**: **,** (and) **;** (or) **:-** (if) **not** (not);
- **Variables**: a variable in Prolog is a string of letters, digits, and underscores (_) beginning either with a capital letter or with an underscore;
- **Facts**: a fact is a predicate expression that makes a declarative statement about the problem domain;
- **Rules**: a rule is a predicate expression that uses logical implication (**:-**) to describe a relationship among facts. Thus a Prolog rule takes the form:

  left_hand_side :- *right_hand_side* .

- **Queries**: the Prolog interpreter answers queries on the facts and rules represented in its database. By asking a question, Prolog is asked whether it can prove that the question is true. If the answer is "yes", Prolog answers "yes" and displays all the links between the variables made to obtain the answer. If it cannot prove that the query is true, it answers "no".;

Furthermore, in Prolog, the technique of backtracking is employed. Backtracking is a process where Prolog examines the truthfulness of various predicates by evaluating their correctness. Essentially, the Prolog interpreter explores different potential paths through backtracking until it discovers a valid route that reaches the final destination. Additionally, there are several built-in predicates, i.e., native rules provided by Prolog, such as "length(?List, ?Length)" which allows obtaining the length of a list. Prolog is so powerful that it allows defining a knowledge base in just a few lines. We leveraged this capability to create our rule-based system for meals and ingredients.

3.2.1 Prolog Implementation

We have implemented the model defined in the DMN file using Prolog and tested our implementation on the site SWISH Prolog[6]. As with Task 1.1, we have also restructured the Prolog file, which now applies restrictions first at the ingredient level and then at the meal level. Consequently, if an ingredient is categorized as vegetarian and a meal contains that ingredient (unless the meal also includes a carnivorous ingredient), the meal itself is classified as vegetarian. The same logic applies for carnivorous ingredients. However, if a meal contains both a carnivorous and a vegetarian ingredient, it is classified as neither carnivorous nor vegetarian, and is therefore considered omnivorous (note that all carnivorous and vegetarian meals are inherently omnivorous). Meals that contain neither carnivorous nor vegetarian ingredients are still visible to both carnivorous and vegetarian users. Our Prolog code is described below.

[5]Prolog Wikipedia: https://it.wikipedia.org/wiki/Prolog
[6]Prolog Testing: https://swish.swi-prolog.org/

This section defines a list of ingredients used in various meals. Each ingredient is declared with the ingredient predicate, which effectively lists all the components available for meal preparation. This comprehensive list covers a broad spectrum of ingredients, including vegetables, meats, dairy products, and staples like flour and oil. It facilitates the subsequent steps of the program, where these ingredients will be associated with their caloric values and categorization.

```prolog
% Define the list of ingredients used in meals
ingredient(salt).
ingredient(oil).
ingredient(carrot).
ingredient(cabbage).
ingredient(puff_pastry).
ingredient(ciauscolo).
ingredient(salame).
ingredient(ham).
ingredient(black_pepper).
ingredient(cow_cheese).
ingredient(sheep_cheese).
ingredient(parmesan).
ingredient(mozzarella_cheese).
ingredient(pasta).
ingredient(egg).
ingredient(pork_cheek).
ingredient(shrimp).
ingredient(mussel).
ingredient(chili).
ingredient(garlic).
ingredient(tomato).
ingredient(bechamel).
ingredient(minced_meat).
ingredient(pork_chop).
ingredient(pork_rib).
ingredient(sausage).
ingredient(steak).
ingredient(calamari).
ingredient(sword_fish).
ingredient(salmon).
ingredient(potato).
ingredient(eggplant).
ingredient(type_0_flour).
ingredient(water).
ingredient(sugar_and_food_coloring).
ingredient(corn).
ingredient(lettuce).
ingredient(apple).
ingredient(banana).
ingredient(strawberry).
```

Listing 3.1: Facts for Ingredients

This section specifies the caloric content for each ingredient using the kcal_ingredient predicate. It maps each ingredient to its caloric value in kilocalories. Defining caloric values is essential for evaluating the nutritional content of meals. The data provided allows for detailed caloric calculations which are crucial for dietary planning and nutritional analysis.

```prolog
% Define the caloric content of each ingredient
kcal_ingredient(salt, 0).
kcal_ingredient(oil, 60).
kcal_ingredient(carrot, 27).
kcal_ingredient(cabbage, 25).
kcal_ingredient(puff_pastry, 150).
kcal_ingredient(ciauscolo, 70).
kcal_ingredient(salame, 50).
kcal_ingredient(ham, 45).
kcal_ingredient(black_pepper, 25).
kcal_ingredient(cow_cheese, 120).
kcal_ingredient(sheep_cheese, 135).
kcal_ingredient(parmesan, 85).
kcal_ingredient(mozzarella_cheese, 120).
kcal_ingredient(pasta, 300).
kcal_ingredient(egg, 80).
kcal_ingredient(pork_cheek, 150).
kcal_ingredient(shrimp, 35).
kcal_ingredient(mussel, 40).
kcal_ingredient(chili, 15).
kcal_ingredient(garlic, 3).
kcal_ingredient(tomato, 30).
kcal_ingredient(bechamel, 105).
kcal_ingredient(minced_meat, 120).
kcal_ingredient(pork_chop, 125).
kcal_ingredient(pork_rib, 70).
kcal_ingredient(sausage, 205).
kcal_ingredient(steak, 485).
kcal_ingredient(calamari, 50).
kcal_ingredient(sword_fish, 120).
kcal_ingredient(salmon, 150).
kcal_ingredient(potato, 450).
kcal_ingredient(eggplant, 18).
kcal_ingredient(type_0_flour, 500).
kcal_ingredient(water, 0).
kcal_ingredient(sugar_and_food_coloring, 465).
kcal_ingredient(corn, 50).
kcal_ingredient(lettuce, 25).
kcal_ingredient(apple, 35).
kcal_ingredient(banana, 40).
kcal_ingredient(strawberry, 22).
```

Listing 3.2: Ingredient/Kcal association

Here are ingredients that are classified as carnivorous. If a meal contains a carnivorous ingredient and no vegetarian ingredients, then that meal is considered carnivorous.

```prolog
% Define which ingredients are carnivorous
ingredient_carnivore(ciauscolo).
ingredient_carnivore(salame).
ingredient_carnivore(ham).
ingredient_carnivore(pork_cheek).
ingredient_carnivore(minced_meat).
ingredient_carnivore(pork_rib).
ingredient_carnivore(sausage).
ingredient_carnivore(shrimp).
ingredient_carnivore(mussel).
ingredient_carnivore(steak).
ingredient_carnivore(calamari).
ingredient_carnivore(sword_fish).
ingredient_carnivore(salmon).
```

Listing 3.3: Carnivorous Ingredient

Here are ingredients that are classified as vegetarian. If a meal contains a vegetarian ingredient and no carnivorous ingredients, then that meal is considered vegetarian.

```
% Define which ingredients are vegetarian
ingredient_vegetarian(carrot).
ingredient_vegetarian(cabbage).
ingredient_vegetarian(eggplant).
ingredient_vegetarian(lettuce).
ingredient_vegetarian(apple).
ingredient_vegetarian(banana).
ingredient_vegetarian(strawberry).
```

Listing 3.4: Vegetarian Ingredient

Here are all the ingredients that contain allergens (gluten or lactose).

```
% Define which ingredients cause lactose intolerance
ingredient_with_lactose_intolerance(cow_cheese).
ingredient_with_lactose_intolerance(sheep_cheese).
ingredient_with_lactose_intolerance(parmesan).
ingredient_with_lactose_intolerance(mozzarella_cheese).
ingredient_with_lactose_intolerance(bechamel).

% Define which ingredients contain gluten
ingredient_with_gluten_intolerance(type_0_flour).
ingredient_with_gluten_intolerance(pasta).
ingredient_with_gluten_intolerance(puff_pastry).
```

Listing 3.5: Ingredients that contains an intolerance

Meals are defined with the meal predicate, specifying the meal name, course type (e.g., appetizer, first_dish), and a list of ingredients. The structured meal definitions provide a foundation for querying and analyzing meal options based on their composition. This organization facilitates the creation of meal recommendations and nutritional assessments.

```
% Define various meals along with their courses and ingredients
meal(involtini_primavera, appetizer, [salt, oil, carrot, cabbage,
    puff_pastry]).
meal(affettati_misti, appetizer, [salt, ciauscolo, salame, ham, black_pepper
    ]).
meal(formaggi_misti, appetizer, [salt, cow_cheese, sheep_cheese, parmesan]).
meal(tagliatelle_alla_marinara, first_dish, [salt, oil, pasta, shrimp,
    mussel]).
meal(rigatoni_alla_carbonara, first_dish, [salt, oil, pasta, sheep_cheese,
    egg, black_pepper, pork_cheek]).
meal(trofie_cacio_e_pepe, first_dish, [salt, oil, pasta, black_pepper,
    sheep_cheese]).
meal(spaghetti_aglio_olio_e_peperoncino, first_dish, [salt, oil, pasta,
    garlic, chili]).
meal(lasagne, first_dish, [salt, oil, carrot, pasta, parmesan, tomato,
    bechamel, minced_meat]).
meal(grigliata_di_maiale_mista, second_course, [salt, pork_rib, sausage,
    pork_chop]).
meal(bistecca_alla_fiorentina, second_course, [salt, oil, steak]).
meal(pesce_arrosto, second_course, [salt, oil, shrimp, mussel, calamari,
    sword_fish, salmon]).
meal(verdure_miste, second_course, [salt, cabbage, eggplant, lettuce]).
meal(patatine_fritte, sidedish, [salt, oil, potato]).
meal(melanzane_grigliate, sidedish, [salt, oil, eggplant]).
meal(insalata_mista, sidedish, [salt, oil, lettuce, corn]).
meal(pizza_margherita, main_dish, [salt, oil, cow_cheese, tomato, water,
    type_0_flour]).
meal(pizza_bianca, main_dish, [salt, oil, water, type_0_flour]).
meal(water, drink, [water]).
meal(coca_cola, drink, [water, sugar_and_food_coloring]).
meal(fruit_salad, dessert, [apple, banana, strawberry]).
```

Listing 3.6: Facts for Meals

These predicates (described below) determine the dietary classification of a meal:

- vegetarian_meal checks if all ingredients are vegetarian or non-carnivorous.
- carnivorous_meal ensures all ingredients are either carnivorous or non-vegetarian.
- omnivore_meal identifies meals containing both carnivorous and vegetarian ingredients.

These predicates enable the categorization of meals according to dietary preferences, assisting users in selecting suitable meal options based on their dietary requirements.

```
% Determine if a meal is vegetarian
vegetarian_meal(Meal, Course) :-
    meal(Meal, Course, Ingredients),
    forall(member(Ingredient, Ingredients),
           (ingredient_vegetarian(Ingredient); \+ ingredient_carnivore(
    Ingredient))).

% Determine if a meal is carnivorous
carnivorous_meal(Meal, Course) :-
    meal(Meal, Course, Ingredients),
    % Ensure all ingredients are either carnivorous or not vegetarian
    forall(member(Ingredient, Ingredients),
           (ingredient_carnivore(Ingredient); \+ ingredient_vegetarian(
    Ingredient))).

% Define meals that contain both carnivorous and vegetarian ingredients
omnivore_meal(Meal, Course) :-
    meal(Meal, Course, Ingredients),
    forall(member(Ingredient, Ingredients), ingredient(Ingredient)).
```

Listing 3.7: Rules for Meal: Category

These predicates identify meals that contain ingredients causing gluten or lactose intolerance. They use findall to collect relevant meal-course pairs and then filter out duplicates. These checks are critical for ensuring that meals meet specific dietary restrictions. They support the customization of meal plans for individuals with food intolerances.

```
% Find meals with gluten intolerance
meal_with_gluten_intolerance(Meal, Course) :-
    findall(Meal-Course,
            (meal(Meal, Course, Ingredients),
             member(Ingredient, Ingredients),
             ingredient_with_gluten_intolerance(Ingredient)),
            MealsWithGlutenIntolerance),
    list_to_set(MealsWithGlutenIntolerance, UniqueMeals),
    member(Meal-Course, UniqueMeals).

% Find meals with lactose intolerance
meal_with_lactose_intolerance(Meal, Course) :-
    findall(Meal-Course,
            (meal(Meal, Course, Ingredients),
             member(Ingredient, Ingredients),
             ingredient_with_lactose_intolerance(Ingredient)),
            MealsWithLactoseIntolerance),
    list_to_set(MealsWithLactoseIntolerance, UniqueMeals),
    member(Meal-Course, UniqueMeals).
```

Listing 3.8: Rules for Meal: Allergy

Description:

- **meal_calories** computes the total caloric content of a meal by summing the calories of its ingredients.
- **calorie_conscious_levels** categorizes meals into calorie-conscious levels based on their total caloric content.

These calculations are essential for dietary management, providing insights into the energy content of meals and helping users make informed choices based on their caloric needs.

```
% Calculate the total caloric content of a meal
meal_calories(Meal, Course, TotalCalories) :-
    meal(Meal, Course, Ingredients),
    findall(Calories,
            (member(Ingredient, Ingredients),
             kcal_ingredient(Ingredient, Kcal),
             Calories is Kcal),
            CaloriesList),
    sum_list(CaloriesList, TotalCalories).

% Determine calorie-conscious levels based on total calories
calorie_conscious_levels(Meal, Course, Levels) :-
    meal_calories(Meal, Course, TotalCalories),
    % Determine the highest applicable level based on total calories
    ( TotalCalories > 650 -> HighestLevel = 0
    ; TotalCalories =< 250 -> HighestLevel = 3
    ; TotalCalories =< 450 -> HighestLevel = 2
    ; TotalCalories =< 650 -> HighestLevel = 1
    ),
    % Generate all levels up to the highest applicable level
    findall(Level, (between(0, HighestLevel, Level)), Levels).
```

Listing 3.9: Rules for compute the total kcals for a Meal and calorie-conscious levels

The **guest_preferences** predicate filters meals based on category (carnivorous, vegetarian, omnivore), calorie level, and allergies. It combines multiple conditions to refine meal recommendations. This predicate provides a robust mechanism for tailoring meal recommendations to individual preferences and dietary restrictions, facilitating personalized meal planning.

```
% Filter meals based on guest preferences, including category, calorie level
    , and allergies
guest_preferences(Category, CalorieLevel, Allergies, Meal, Course) :-
    % Filtered meals by Category (carnivorous, vegetarian, omnivore)
    ( Category = carnivorous -> carnivorous_meal(Meal, Course)
    ; Category = vegetarian -> vegetarian_meal(Meal, Course)
    ; Category = omnivore -> omnivore_meal(Meal, Course)
    ),
    % Filtered meals by calorie_consciou_level
    calorie_conscious_levels(Meal, Course, Levels),
    member(CalorieLevel, Levels),
    % Filtered meals by allergies
    ( Allergies = none -> true
    ; ( Allergies = lactose -> not(meal_with_lactose_intolerance(Meal,
    Course))
      ; Allergies = gluten -> not(meal_with_gluten_intolerance(Meal, Course)
    )
    )
    ).
```

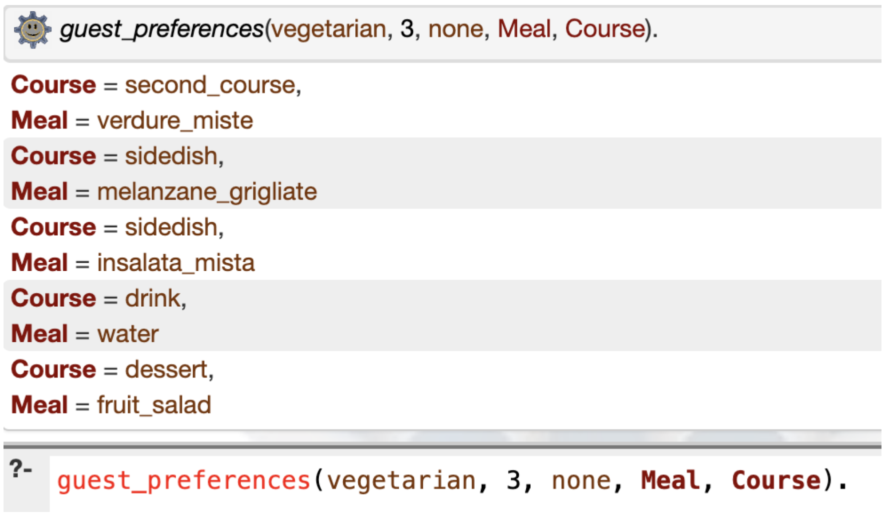
Listing 3.10: Rule for Guest Preferences

This Prolog code offers a comprehensive system for managing meal ingredients, analyzing their nutritional content, and accommodating dietary preferences and restrictions. The organization of predicates ensures flexibility and accuracy in meal recommendations based on various criteria.

Prolog Testing

We want to illustrate some tests that we have done to see if the query of guest preferences woks well.

In this scenario, if the user is vegetarian, has a moderate to high calorie awareness



```
guest_preferences(vegetarian, 3, none, Meal, Course).

Course = second_course,
Meal = verdure_miste
Course = sidedish,
Meal = melanzane_grigliate
Course = sidedish,
Meal = insalata_mista
Course = drink,
Meal = water
Course = dessert,
Meal = fruit_salad

?- guest_preferences(vegetarian, 3, none, Meal, Course).
```

Figure 3.9: Prolog Testing 1

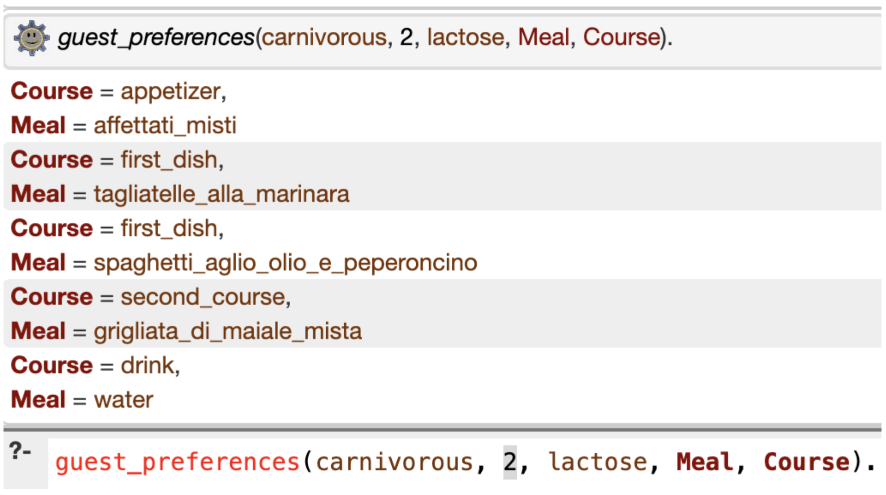
and has no allergies, the system returns all the dishes and the relative courses.

In the other case described below, if the user is carnivorous, has a moderate to low calorie awareness per meal, and it has a lactose intolerance, the following meals are returned: 'affettati_misti', 'spaghetti_aglio_olio_e_peperoncino', 'tagliatelle_alla_marinara', 'grigliata_di_magliale_mista', 'water'.

Here, we provide example queries to run in the simulator described before for other results.

```
% guest_preferences(vegetarian, 3, none, Meal, second_course).

% guest_preferences(carnivorous, 1, none, Meal, Course).

% guest_preferences(omnivore, 0, gluten, Meal, Course).
```

Listing 3.11: Queries for Prolog Testing



```
guest_preferences(carnivorous, 2, lactose, Meal, Course).

Course = appetizer,
Meal = affettati_misti
Course = first_dish,
Meal = tagliatelle_alla_marinara
Course = first_dish,
Meal = spaghetti_aglio_olio_e_peperoncino
Course = second_course,
Meal = grigliata_di_maiale_mista
Course = drink,
Meal = water

?- guest_preferences(carnivorous, 2, lactose, Meal, Course).
```

Figure 3.10: Prolog Testing 2

3.3 Ontology Engineering

In Computer Science, ontology engineering is a discipline focused on methodologies for constructing ontologies. This process involves formally representing and defining categories, properties, and relationships among concepts, data, and entities. Ontologies can be structured using various data models such as RDF, OWL, Neo4J, and others. For our ontology construction, we employed RDF.

RDF

The Resource Description Framework (**RDF**) is a standardized approach utilized for describing and exchanging graph data in a general manner. RDF enables the description of resources that are utilized in constructing knowledge graphs.
A **Knowledge Graph** depicts a network of real-world entities such as objects, events, situations, or concepts, highlighting the relationships that exist between them.

Protégé

Protégé is a freely available, open-source ontology editor and knowledge management system. It enables the definition of Semantic Web Rule Language (**SWRL**) to establish inference rules that generate new data from existing ontologies. Additionally, Protégé supports the Sparql Protocol And Rdf Query Language (**SPARQL**), a semantic query language designed for databases capable of retrieving and manipulating data stored in RDF format.

3.3.1 Our Ontology

Using Protégé, we have developed our ontology by incorporating the primary entities. Compared to the initial submission in June, we have made the following changes: many of the SPARQL queries have been included in the report and elaborated upon in detail. The SHACL file has also been thoroughly explained and modified to ensure the SHACL constraints do not appear artificial. We have eliminated redundancies, and for ingredients, we added the Object Property food_hasCategory, which replaces the

Object Property meal_hasCategory. This change makes the property available not only at the subclass level of Food (i.e., Meal) but also for the subclass Ingredient.

Regarding SWRL, we have implemented the required modifications, and by adding two additional Object Properties (meal_notContainsGlutenIntolerance and meal_notContainsLactoseIntolerance), it is now possible to infer the meals that a user can eat based on their allergy type (if any), category, calorie-consciousness, and course preference.

Classes

The primary entities defined earlier also serve as our main classes. Additionally, we have established a hierarchy to facilitate the future inclusion of other resources, such as potential allergens, types of ingredients and more.

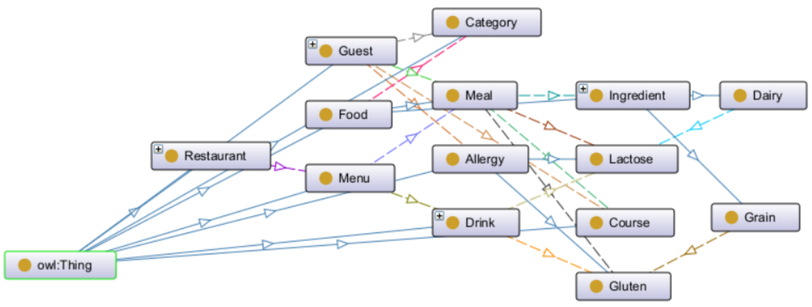


Figure 3.11: Our Ontology Graph in Protégé

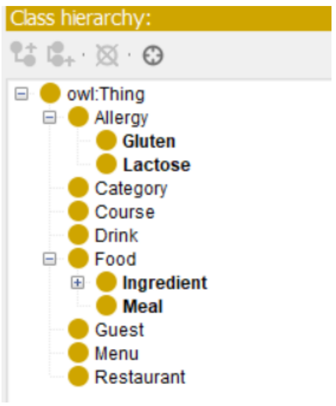


Figure 3.12: Our Ontology Model in Protégé

Object Properties

Object properties are used to represent the relationships between classes. We defined several object properties for the classes:

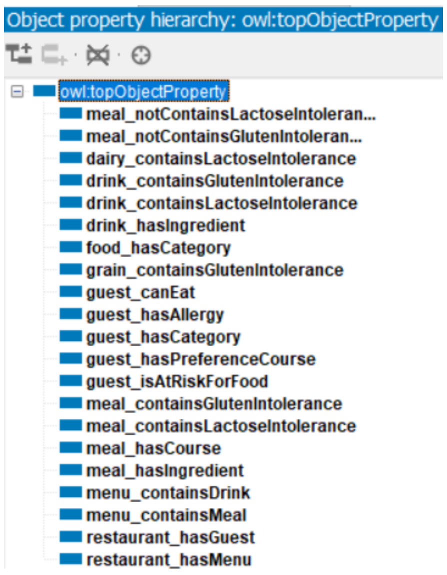
- **restaurant_hasMenu**: to associate a menu with a restaurant;
- **restaurant_hasGuest**: to associate guest with a restaurant;
- **menu_containsDrink**:to associate a drink with a menu;

- **menu_containsMeal**: to associate a meal with a menu;
- **guest_hasAllergies**: it represents a guest that has an allergy;
- **guest_hasCategory**: to associate a category (e.g. Carnivorous) with a guest;
- **guest_hasPreferenceCourse**: to associate a specific course with a guest;
- **guest_isAtRiskForFood**: this is a SWRL rule inferred by the reasoner, which is useful for identifying potential meal-related risks due to allergies;
- **food_hasCategory**: to associate a category with the food (ingredients or meals);
- **meal_containsGlutenIntolerance**: this is an SWRL rule inferred by the reasoner, which identifies if the meal contains ingredients with gluten (e.g., grain);
- **meal_containsLactoseIntolerance**: this is an SWRL rule inferred by the reasoner, which identifies if the meal contains ingredients with lactose (e.g., dairy);
- **meal_notContainsLactoseIntolerance**: added to check if a meal is lactose-free;
- **meal_notContainsGlutenIntolerance**: added to check if a meal is gluten-free;
- **meal_hasCourse**: to associate a course with the meal;
- **meal_hasIngredient**: to associate an ingredient with a meal;
- **drink_hasIngredient**: to associate an ingredient with a drink;
- **drink_containsGlutenIntolerance**: to associate a drink with gluten intolerance;
- **drink_containsLactoseIntolerance**: to associate a drink with lactose;
- **dairy_containsLactoseIntolerance**; to associate the lactose intolerance to class dairy;
- **grain_containsGlutenIntolerance**: to associate the gluten intolerance to class grain.

Data Properties

Data Properties represent the attributes of the classes. Below, all the Data Properties are illustrated:

- **thing_hasName**: this is an attribute that all subclasses of the class Thing possess. It is a string used to identify the name of the object;
- **food_hasKcal**: this is an attribute that all subclasses of the class Food (in this case we have only Ingredient and Meal) posses. It is an integer used to identify the kcal of the food;
- **guest_hasLevelOfCalorieConscious**: this is an attribute for the class Guest that represents the level of calorie-consciousness of a guest. It is an integer ranging between 0 and 3;

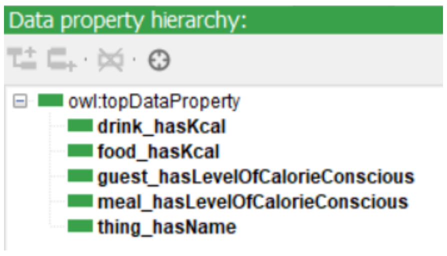


Object property hierarchy: owl:topObjectProperty

owl:topObjectProperty
- meal_notContainsLactoseIntoleran...
- meal_notContainsGlutenIntoleran...
- dairy_containsLactoseIntolerance
- drink_containsGlutenIntolerance
- drink_containsLactoseIntolerance
- drink_hasIngredient
- food_hasCategory
- grain_containsGlutenIntolerance
- guest_canEat
- guest_hasAllergy
- guest_hasCategory
- guest_hasPreferenceCourse
- guest_isAtRiskForFood
- meal_containsGlutenIntolerance
- meal_containsLactoseIntolerance
- meal_hasCourse
- meal_hasIngredient
- menu_containsDrink
- menu_containsMeal
- restaurant_hasGuest
- restaurant_hasMenu

Figure 3.13: Object Properties for our Model

- **meal_hasLevelOfCalorieConscious**: this is an attribute for the class Meal that represents the level of calorie-consciousness of a meal. It is an integer ranging between 0 and 3;
- **drink_hasKcal**: this is an attribute that the class Drink possesses. It is an integer used to identify the kcal of the drink;



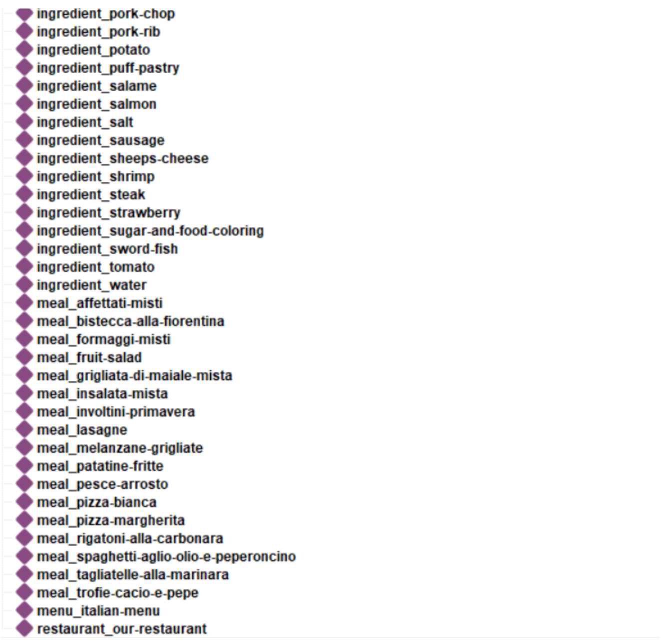
Data property hierarchy:

owl:topDataProperty
- drink_hasKcal
- food_hasKcal
- guest_hasLevelOfCalorieConscious
- meal_hasLevelOfCalorieConscious
- thing_hasName

Figure 3.14: Data Properties for our Model

Individuals

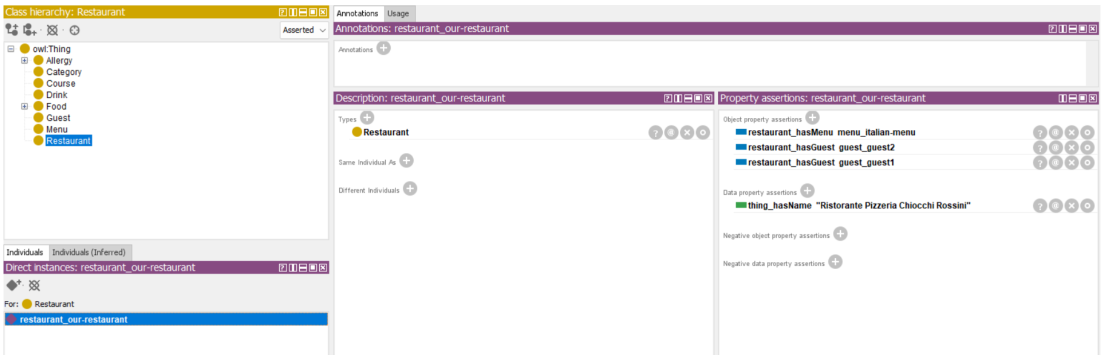
Given the large number of individuals, we will only specify a few examples.



ingredient_pork-chop
ingredient_pork-rib
ingredient_potato
ingredient_puff-pastry
ingredient_salame
ingredient_salmon
ingredient_salt
ingredient_sausage
ingredient_sheeps-cheese
ingredient_shrimp
ingredient_steak
ingredient_strawberry
ingredient_sugar-and-food-coloring
ingredient_sword-fish
ingredient_tomato
ingredient_water
meal_affettati-misti
meal_bistecca-alla-fiorentina
meal_formaggi-misti
meal_fruit-salad
meal_grigliata-di-maiale-mista
meal_insalata-mista
meal_involtini-primavera
meal_lasagne
meal_melanzane-grigliate
meal_patatine-fritte
meal_pesce-arrosto
meal_pizza-bianca
meal_pizza-margherita
meal_rigatoni-alla-carbonara
meal_spaghetti-aglio-olio-e-peperoncino
meal_tagliatelle-alla-marinara
meal_trofie-cacio-e-pepe
menu_italian-menu
restaurant_our-restaurant

Figure 3.15: Some Individuals in Our Model

For the class Restaurant, there is only one instance: restaurant_our-restaurant. It is as follows:



Class hierarchy: Restaurant
Asserted
owl:Thing
Allergy
Category
Course
Drink
Food
Guest
Menu
Restaurant

Individuals | Individuals (Inferred)
Direct instances: restaurant_our-restaurant
For: Restaurant
restaurant_our-restaurant

Annotations | Usage
Annotations: restaurant_our-restaurant
Annotations

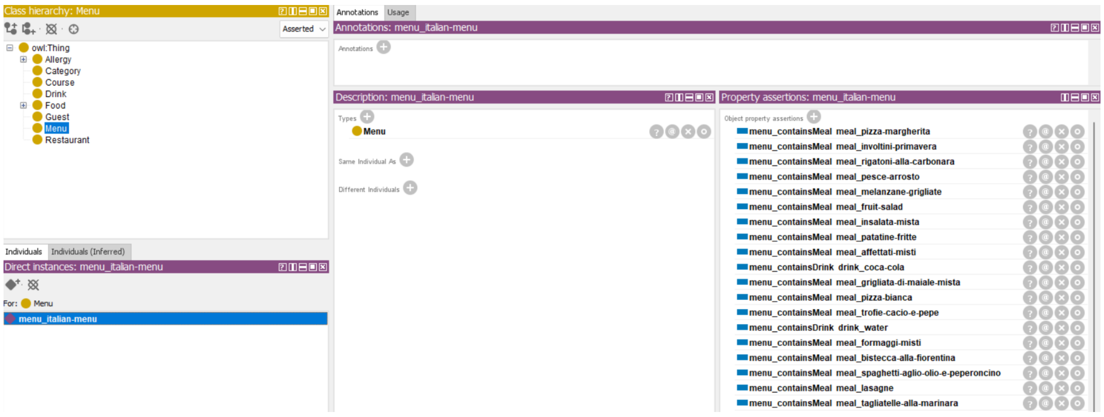
Description: restaurant_our-restaurant
Types
Restaurant
Same Individual As
Different Individuals

Property assertions: restaurant_our-restaurant
Object property assertions
restaurant_hasMenu menu_italian-menu
restaurant_hasGuest guest_guest2
restaurant_hasGuest guest_guest1
Data property assertions
thing_hasName "Ristorante Pizzeria Chiocchi Rossini"
Negative object property assertions
Negative data property assertions

Figure 3.16: Individual Restaurant in Our Model

As can be seen, it has the following characteristics: it includes a menu and two guests, and it has a name.

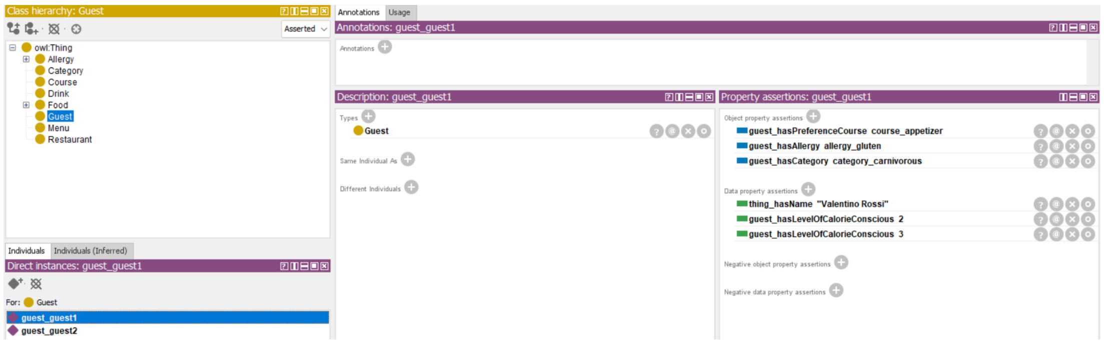
For the Menu class, there is a single instance that includes both drinks and meals:



```
Class hierarchy: Menu
Asserted
owl:Thing
  Allergy
  Category
  Course
  Drink
  Food
  Guest
  Menu
  Restaurant

Individuals | Individuals (Inferred)
Direct instances: menu_italian-menu
For: Menu
menu_italian-menu

Annotations | Usage
Annotations: menu_italian-menu
Annotations

Description: menu_italian-menu
Types
  Menu
Same Individual As
Different Individuals

Property assertions: menu_italian-menu
Object property assertions
  menu_containsMeal  meal_pizza-margherita
  menu_containsMeal  meal_involtini-primavera
  menu_containsMeal  meal_rigatoni-alla-carbonara
  menu_containsMeal  meal_pesce-arrosto
  menu_containsMeal  meal_melanzane-grigliate
  menu_containsMeal  meal_fruit-salad
  menu_containsMeal  meal_insalata-mista
  menu_containsMeal  meal_patatine-fritte
  menu_containsMeal  meal_affettati-misti
  menu_containsDrink  drink_coca-cola
  menu_containsMeal  meal_grigliata-di-maiale-mista
  menu_containsMeal  meal_pizza-bianca
  menu_containsMeal  meal_trofie-cacio-e-pepe
  menu_containsDrink  drink_water
  menu_containsMeal  meal_formaggi-misti
  menu_containsMeal  meal_bistecca-alla-fiorentina
  menu_containsMeal  meal_spaghetti-aglio-olio-e-peperoncino
  menu_containsMeal  meal_lasagne
  menu_containsMeal  meal_tagliatelle-alla-marinara
```

Figure 3.17: Individual Menu in Our Model

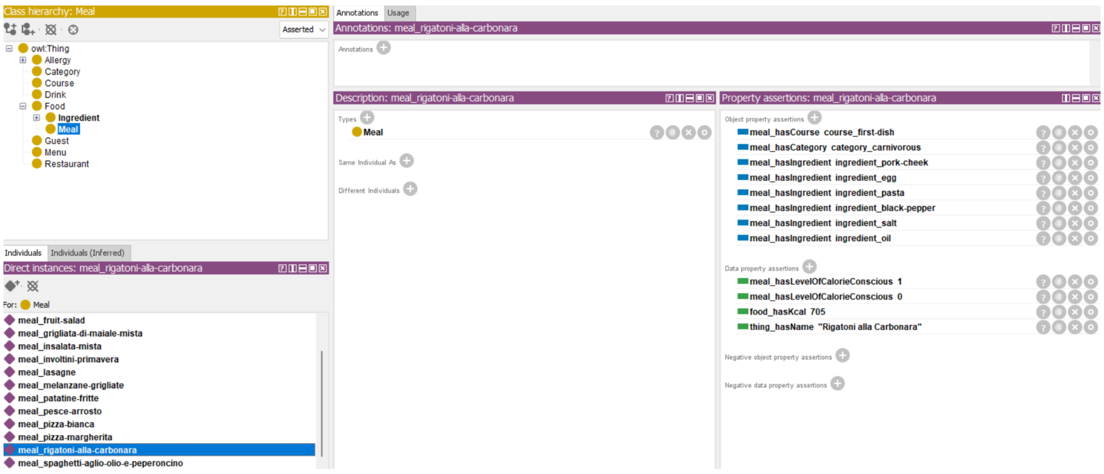
Regarding the Guest class, we have two individuals. The following figure depicts guest1:



```
Class hierarchy: Guest
Asserted
owl:Thing
  Allergy
  Category
  Course
  Drink
  Food
  Guest
  Menu
  Restaurant

Individuals | Individuals (Inferred)
Direct instances: guest_guest1
For: Guest
guest_guest1
guest_guest2

Annotations | Usage
Annotations: guest_guest1
Annotations

Description: guest_guest1
Types
  Guest
Same Individual As
Different Individuals

Property assertions: guest_guest1
Object property assertions
  guest_hasPreferenceCourse  course_appetizer
  guest_hasAllergy  allergy_gluten
  guest_hasCategory  category_carnivorous
Data property assertions
  thing_hasName  "Valentino Rossi"
  guest_hasLevelOfCalorieConscious  2
  guest_hasLevelOfCalorieConscious  3
Negative object property assertions
Negative data property assertions
```

Figure 3.18: Individual Guest1 in Our Model

For the Meal class, there are 17 individuals. Below is an example of one individual: rigatoni_alla_carbonara.



```
Class hierarchy: Meal
Asserted
owl:Thing
  Allergy
  Category
  Course
  Drink
  Food
    Ingredient
    Meal
  Guest
  Menu
  Restaurant

Individuals | Individuals (Inferred)
Direct instances: meal_rigatoni-alla-carbonara
For: Meal
meal_fruit-salad
meal_grigliata-di-maiale-mista
meal_insalata-mista
meal_involtini-primavera
meal_lasagne
meal_melanzane-grigliate
meal_patatine-fritte
meal_pesce-arrosto
meal_pizza-bianca
meal_pizza-margherita
meal_rigatoni-alla-carbonara
meal_spaghetti-aglio-olio-e-peperoncino

Annotations | Usage
Annotations: meal_rigatoni-alla-carbonara
Annotations

Description: meal_rigatoni-alla-carbonara
Types
  Meal
Same Individual As
Different Individuals

Property assertions: meal_rigatoni-alla-carbonara
Object property assertions
  meal_hasCourse  course_first-dish
  meal_hasCategory  category_carnivorous
  meal_hasIngredient  ingredient_pork-cheek
  meal_hasIngredient  ingredient_egg
  meal_hasIngredient  ingredient_pasta
  meal_hasIngredient  ingredient_black-pepper
  meal_hasIngredient  ingredient_salt
  meal_hasIngredient  ingredient_oil
Data property assertions
  meal_hasLevelOfCalorieConscious  1
  meal_hasLevelOfCalorieConscious  0
  food_hasKcal  705
  thing_hasName  "Rigatoni alla Carbonara"
Negative object property assertions
Negative data property assertions
```

Figure 3.19: Individual Rigatoni alla Carbonara in Our Model

3.3.2 SWRL Rules

We have exploited the reasoner provided by Protégé its power by defining the following rules:

GuestIsAtRiskForFoodGluten

$$\begin{aligned}&\text{Guest}(?guest) \wedge \text{guest\_hasAllergy}(?guest, ?allergy)\\ &\quad\wedge \text{Meal}(?meal) \wedge \text{meal\_containsGlutenIntolerance}(?meal, ?allergy)\\ &\quad\rightarrow \text{guest\_isAtRiskForFood}(?guest, ?meal)\end{aligned}$$

This rule is designed to infer meals that pose a risk to the user based on whether the guest has an allergy, in this case gluten intolerance, or not, and to display them accordingly.

GuestIsAtRiskForFoodLactose

$$\begin{aligned}&\text{Guest}(?guest) \wedge \text{guest\_hasAllergy}(?guest, ?allergy)\\ &\quad\wedge \text{Meal}(?meal) \wedge \text{meal\_containsLactoseIntolerance}(?meal, ?allergy)\\ &\quad\rightarrow \text{guest\_isAtRiskForFood}(?guest, ?meal)\end{aligned}$$

This rule is designed to infer meals that pose a risk to the user based on whether the guest has an allergy, in this case lactose intolerance, or not, and to display them accordingly.

MealContainsGlutenIntolerance

$$\begin{aligned}&\text{Meal}(?meal) \wedge \text{meal\_hasIngredient}(?meal, ?ingredient)\\ &\quad\wedge \text{Ingredient}(?ingredient) \wedge \text{grain\_containsGlutenIntolerance}(?ingredient, ?gluten)\\ &\rightarrow \text{meal\_containsGlutenIntolerance}(?meal, ?gluten)\end{aligned}$$

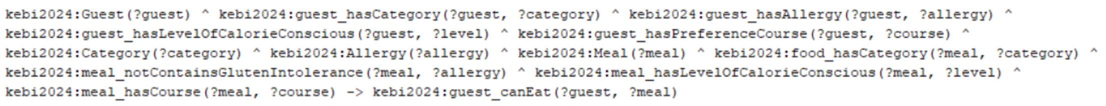
This rule is used to infer meals that contain ingredients which include a gluten intolerance. Therefore, the reasoning is: if the meal contains an ingredient that has a gluten intolerance, then the meal itself has a gluten intolerance.

MealContainsLactoseIntolerance

$$\begin{aligned}&\text{Meal}(?meal) \wedge \text{meal\_hasIngredient}(?meal, ?ingredient)\\ &\quad\wedge \text{Ingredient}(?ingredient) \wedge \text{dairy\_containsLactoseIntolerance}(?ingredient, ?gluten)\\ &\rightarrow \text{meal\_containsLactoseIntolerance}(?meal, ?gluten)\end{aligned}$$

This rule is used to infer meals that contain ingredients which include a lactose intolerance. Therefore, the reasoning is: if the meal contains an ingredient that has a lactose intolerance, then the meal itself has a lactose intolerance.

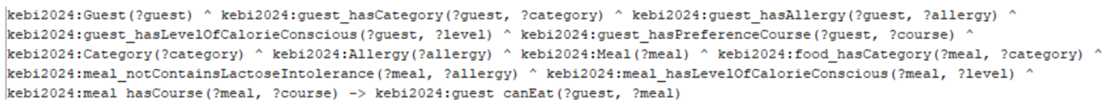
Rule for inferring meals to eat - gluten intolerance This rule is used to infer the types of meals a guest can eat based on their preferences and whether they have a gluten intolerance.



```
kebi2024:Guest(?guest) ^ kebi2024:guest_hasCategory(?guest, ?category) ^ kebi2024:guest_hasAllergy(?guest, ?allergy) ^
kebi2024:guest_hasLevelOfCalorieConscious(?guest, ?level) ^ kebi2024:guest_hasPreferenceCourse(?guest, ?course) ^
kebi2024:Category(?category) ^ kebi2024:Allergy(?allergy) ^ kebi2024:Meal(?meal) ^ kebi2024:food_hasCategory(?meal, ?category) ^
kebi2024:meal_notContainsGlutenIntolerance(?meal, ?allergy) ^ kebi2024:meal_hasLevelOfCalorieConscious(?meal, ?level) ^
kebi2024:meal_hasCourse(?meal, ?course) -> kebi2024:guest_canEat(?guest, ?meal)
```

Figure 3.20: Rule for inferring meals to eat - gluten intolerance

Rule for inferring meals to eat - lactose intolerance This rule is used to infer the types of meals a guest can eat based on their preferences and whether they have a lactose intolerance.



```
kebi2024:Guest(?guest) ^ kebi2024:guest_hasCategory(?guest, ?category) ^ kebi2024:guest_hasAllergy(?guest, ?allergy) ^
kebi2024:guest_hasLevelOfCalorieConscious(?guest, ?level) ^ kebi2024:guest_hasPreferenceCourse(?guest, ?course) ^
kebi2024:Category(?category) ^ kebi2024:Allergy(?allergy) ^ kebi2024:Meal(?meal) ^ kebi2024:food_hasCategory(?meal, ?category) ^
kebi2024:meal_notContainsLactoseIntolerance(?meal, ?allergy) ^ kebi2024:meal_hasLevelOfCalorieConscious(?meal, ?level) ^
kebi2024:meal hasCourse(?meal, ?course) -> kebi2024:guest canEat(?guest, ?meal)
```

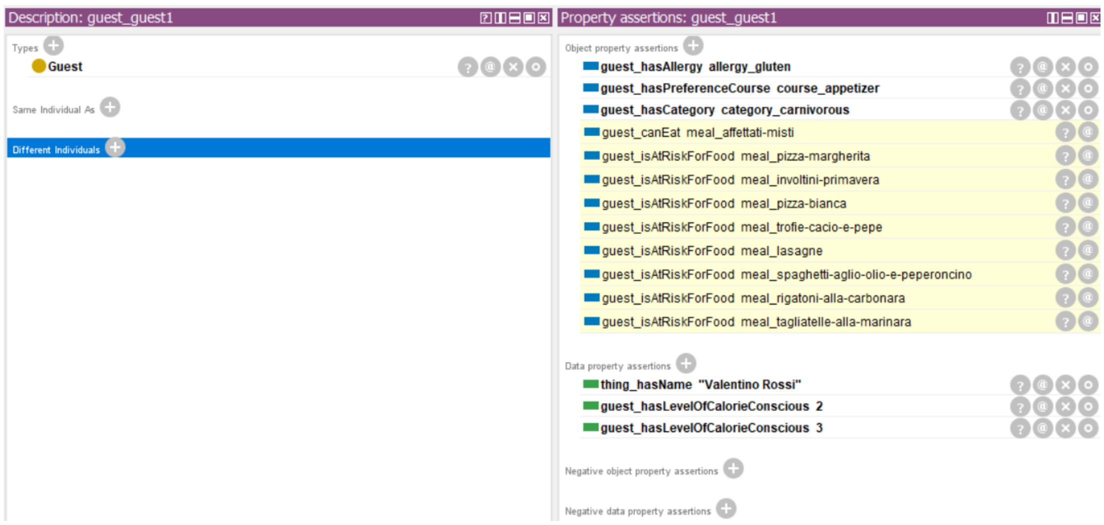
Figure 3.21: Rule for inferring meals to eat - lactose intolerance

Unfortunately, it was not possible to further infer the data property field guest_hasLevelOfCalorieConscious and the field meal_hasLevelOfCalorieConscious with Reasoner HermiT 1.4.3.456 in a short time because it does not support SWRL built-in atoms. We did attempt to use the Pellet (incremental) Reasoner; however, sometimes it worked correctly and other times it resulted in errors. Therefore, we left the two fields without inferences, opting for a more rudimentary but still functional approach.

Finally, the lack of support for negation in SWRL impacted the construction of our model (note the addition of the two Object Properties, meal_notContainsGlutenIntolerance and meal_notContainsLactoseIntolerance).

SWRL Testing

Below are examples of SWRL rule testing executed using Reasoner HermiT 1.4.3.456:



Description: guest_guest1

Types
Guest

Same Individual As

Different Individuals

Property assertions: guest_guest1

Object property assertions
guest_hasAllergy allergy_gluten
guest_hasPreferenceCourse course_appetizer
guest_hasCategory category_carnivorous
guest_canEat meal_affettati-misti
guest_isAtRiskForFood meal_pizza-margherita
guest_isAtRiskForFood meal_involtini-primavera
guest_isAtRiskForFood meal_pizza-bianca
guest_isAtRiskForFood meal_trofie-cacio-e-pepe
guest_isAtRiskForFood meal_lasagne
guest_isAtRiskForFood meal_spaghetti-aglio-olio-e-peperoncino
guest_isAtRiskForFood meal_rigatoni-alla-carbonara
guest_isAtRiskForFood meal_tagliatelle-alla-marinara

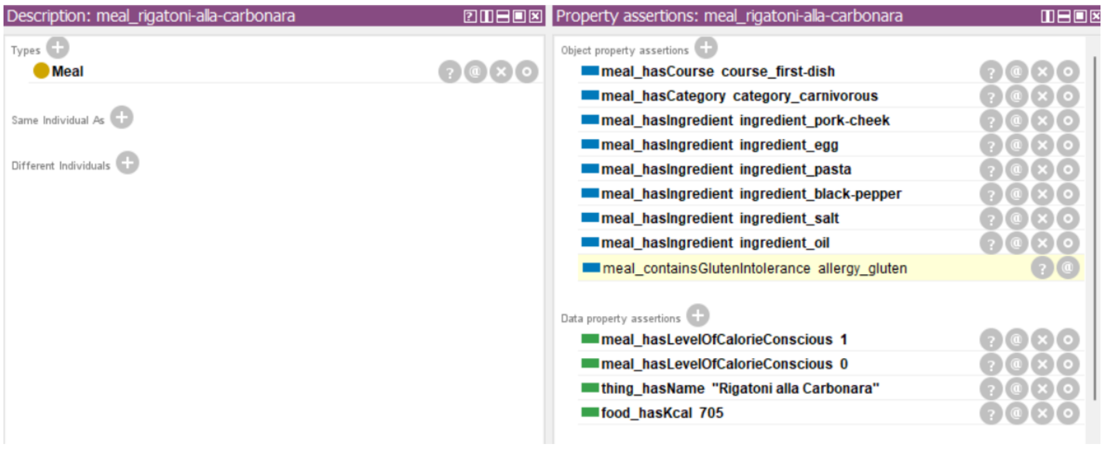
Data property assertions
thing_hasName "Valentino Rossi"
guest_hasLevelOfCalorieConscious 2
guest_hasLevelOfCalorieConscious 3

Negative object property assertions

Negative data property assertions

Figure 3.22: SWRL Testing 1 - Guest 1: Suggesting Meals

As depicted in the first figure (Figure 3.22), the high-risk dishes for the user are immediately inferred, and the food that guest1 can safely consume (affettati misti) is correctly identified.



Description: meal_rigatoni-alla-carbonara

Types
Meal

Same Individual As

Different Individuals

Property assertions: meal_rigatoni-alla-carbonara

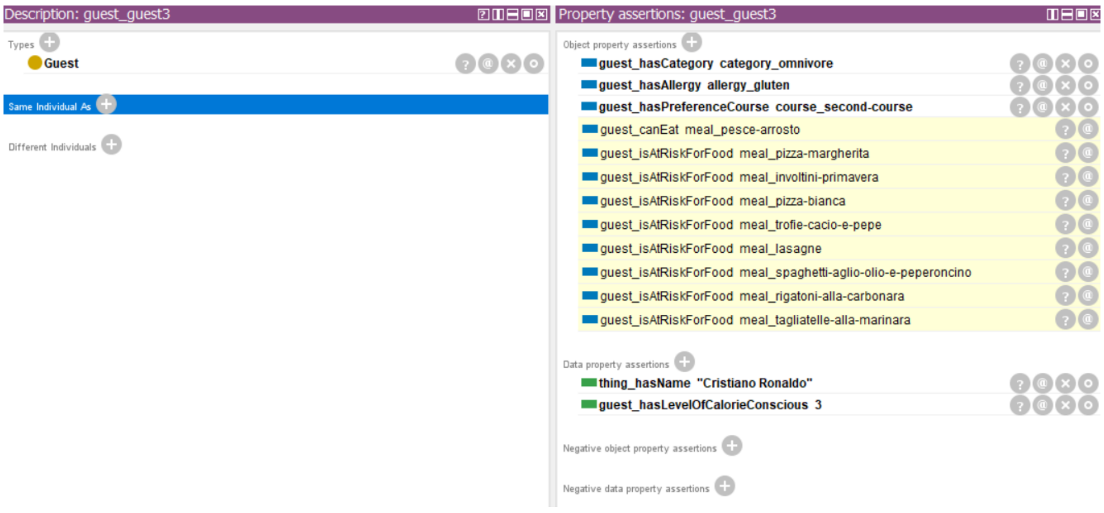
Object property assertions
meal_hasCourse course_first-dish
meal_hasCategory category_carnivorous
meal_hasIngredient ingredient_pork-cheek
meal_hasIngredient ingredient_egg
meal_hasIngredient ingredient_pasta
meal_hasIngredient ingredient_black-pepper
meal_hasIngredient ingredient_salt
meal_hasIngredient ingredient_oil
meal_containsGlutenIntolerance allergy_gluten

Data property assertions
meal_hasLevelOfCalorieConscious 1
meal_hasLevelOfCalorieConscious 0
thing_hasName "Rigatoni alla Carbonara"
food_hasKcal 705

Figure 3.23: SWRL Testing 2 - Meal: Rigatoni all Carbonara

In this case, the inference is based on whether the meal contains an "allergenic" ingredient or not.

Below is the recommended meal inference for the guest based on their preferences: in this case, if the guest is omnivorous, highly calorie-conscious, has a gluten intolerance, and prefers the second course, roasted fish is recommended.



Description: guest_guest3

Types
Guest

Same Individual As

Different Individuals

Property assertions: guest_guest3

Object property assertions
guest_hasCategory category_omnivore
guest_hasAllergy allergy_gluten
guest_hasPreferenceCourse course_second-course
guest_canEat meal_pesce-arrosto
guest_isAtRiskForFood meal_pizza-margherita
guest_isAtRiskForFood meal_involtini-primavera
guest_isAtRiskForFood meal_pizza-bianca
guest_isAtRiskForFood meal_trofie-cacio-e-pepe
guest_isAtRiskForFood meal_lasagne
guest_isAtRiskForFood meal_spaghetti-aglio-olio-e-peperoncino
guest_isAtRiskForFood meal_rigatoni-alla-carbonara
guest_isAtRiskForFood meal_tagliatelle-alla-marinara

Data property assertions
thing_hasName "Cristiano Ronaldo"
guest_hasLevelOfCalorieConscious 3

Negative object property assertions

Negative data property assertions

Figure 3.24: SWRL Testing 3 - Guest 3 suggestion meals

3.3.3 SPARQL Query

Regarding the SPARQL queries, we have developed these queries to filter and retrieve specific information, and subsequently tested them on both GraphDB and Protégé.
SPARQL (SPARQL Protocol and RDF Query Language) is a powerful query language and protocol used for accessing and manipulating data stored in RDF format. It allows users to write queries to extract information from RDF graphs, perform complex searches, and integrate data from diverse sources. SPARQL queries enable precise data

retrieval, making it an essential tool for working with semantic web data and linked data applications.
In the following subsection, we will describe some of the queries found in the file containing all the created queries, queries_sparql.txt[7].

SPARQL Queries Examples

Below is the prefix to be added for the correct execution of the queries.

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/
    kebi2024#>
```

Listing 3.12: Prefixes for all SPARQL Queries

- **Query for Carnivorous**: The following SPARQL query is designed to retrieve all instances of meals that belong to the category "carnivorous" from the dataset.

  ```
  SELECT ?meal
  WHERE {
    ?meal rdf:type kebi:Meal .
    ?meal kebi:food_hasCategory kebi:category_carnivorous .
  }
  ```

  Listing 3.13: Query for retrieve all meals labeled as Carnivorous

  Executing this query on both Protégé and GraphDB yields the results shown in the figures.

- **Query for retrieve all the ingredients in a meal**: This query is used for see all the ingredients in a specific meal, in this case "pizza_margherita".

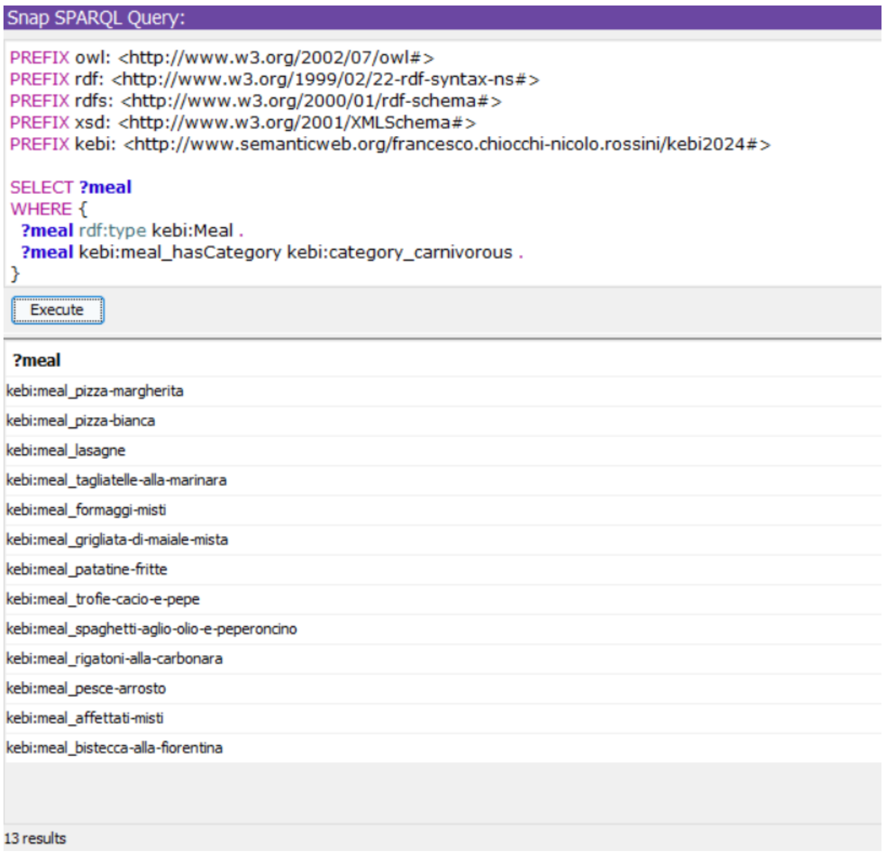
  ```
  SELECT ?ingredient
  WHERE {
    kebi:meal_pizza-margherita kebi:meal_hasIngredient ?ingredient .
  }
  ```

  Listing 3.14: Query for retrieve all the ingredients in a specific meal

  Results:

- **Suggested Meals**: Advanced query for see the meals in base of the preference of the guest, in this case guest1.
  Guest1 Properties:

[7]SPARQL example queries: https://github.com/FrancescoChiocchi8/KEBI-Project/blob/main/Task1/ontology/queries_sparql.txt

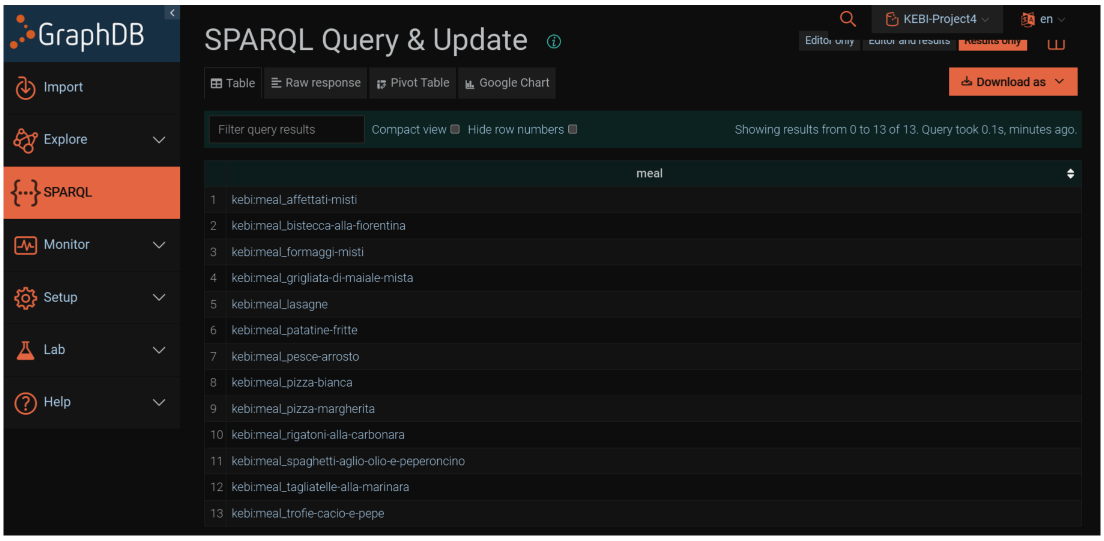

Snap SPARQL Query:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#>

SELECT ?meal
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:meal_hasCategory kebi:category_carnivorous .
}
```

?meal
kebi:meal_pizza-margherita
kebi:meal_pizza-bianca
kebi:meal_lasagne
kebi:meal_tagliatelle-alla-marinara
kebi:meal_formaggi-misti
kebi:meal_grigliata-di-maiale-mista
kebi:meal_patatine-fritte
kebi:meal_trofie-cacio-e-pepe
kebi:meal_spaghetti-aglio-olio-e-peperoncino
kebi:meal_rigatoni-alla-carbonara
kebi:meal_pesce-arrosto
kebi:meal_affettati-misti
kebi:meal_bistecca-alla-fiorentina

13 results

Figure 3.25: SPARQL Query with Protégé: Carnivorous

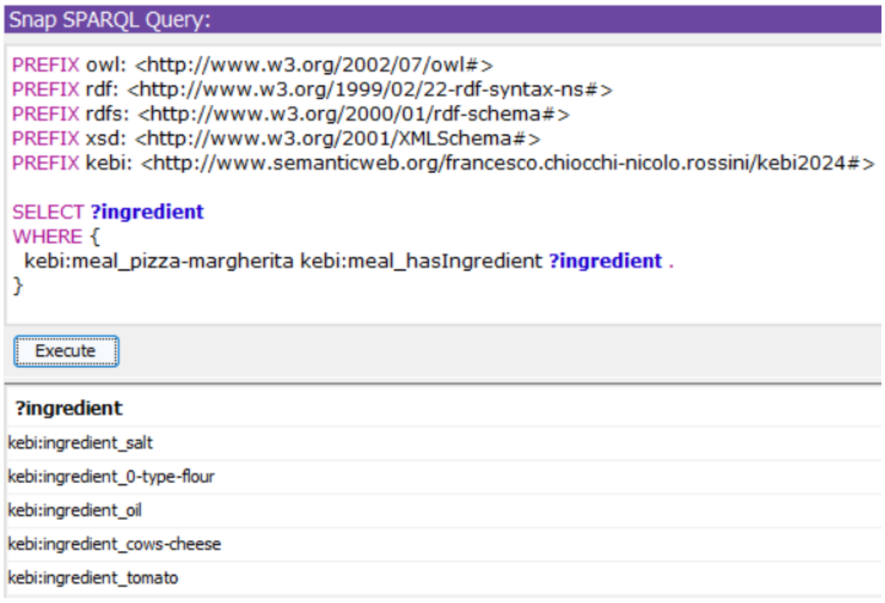


SPARQL Query & Update

Showing results from 0 to 13 of 13. Query took 0.1s, minutes ago.

	meal
1	kebi:meal_affettati-misti
2	kebi:meal_bistecca-alla-fiorentina
3	kebi:meal_formaggi-misti
4	kebi:meal_grigliata-di-maiale-mista
5	kebi:meal_lasagne
6	kebi:meal_patatine-fritte
7	kebi:meal_pesce-arrosto
8	kebi:meal_pizza-bianca
9	kebi:meal_pizza-margherita
10	kebi:meal_rigatoni-alla-carbonara
11	kebi:meal_spaghetti-aglio-olio-e-peperoncino
12	kebi:meal_tagliatelle-alla-marinara
13	kebi:meal_trofie-cacio-e-pepe

Figure 3.26: SPARQL Query with GraphDB: Carnivorous

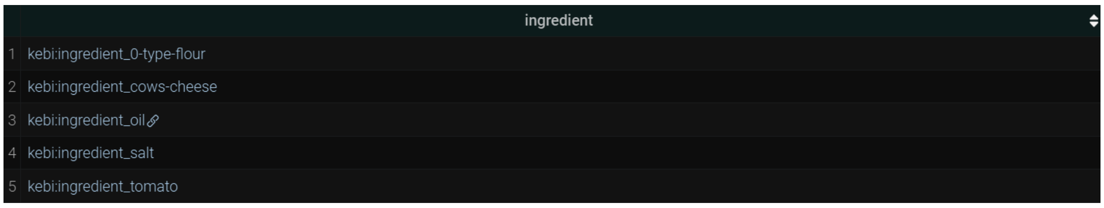


Snap SPARQL Query:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#>

SELECT ?ingredient
WHERE {
 kebi:meal_pizza-margherita kebi:meal_hasIngredient ?ingredient .
}
```

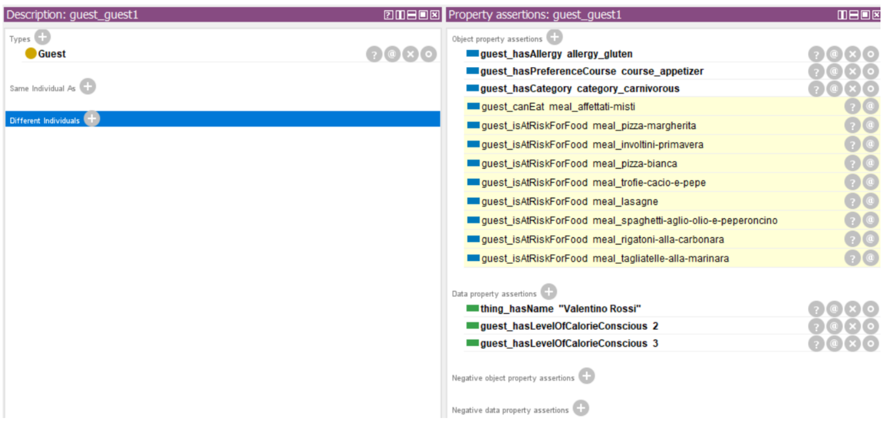
?ingredient
kebi:ingredient_salt
kebi:ingredient_0-type-flour
kebi:ingredient_oil
kebi:ingredient_cows-cheese
kebi:ingredient_tomato

Figure 3.27: SPARQL Query with Protégé: Ingredients in a Meal



	ingredient
1	kebi:ingredient_0-type-flour
2	kebi:ingredient_cows-cheese
3	kebi:ingredient_oil
4	kebi:ingredient_salt
5	kebi:ingredient_tomato

Figure 3.28: SPARQL Query with GraphDB: Ingredients in a Meal



Description: guest_guest1

Types
- Guest

Same Individual As

Different Individuals

Property assertions: guest_guest1

Object property assertions
- guest_hasAllergy allergy_gluten
- guest_hasPreferenceCourse course_appetizer
- guest_hasCategory category_carnivorous
- guest_canEat meal_affettati-misti
- guest_isAtRiskForFood meal_pizza-margherita
- guest_isAtRiskForFood meal_involtini-primavera
- guest_isAtRiskForFood meal_pizza-bianca
- guest_isAtRiskForFood meal_trofie-cacio-e-pepe
- guest_isAtRiskForFood meal_lasagne
- guest_isAtRiskForFood meal_spaghetti-aglio-olio-e-peperoncino
- guest_isAtRiskForFood meal_rigatoni-alla-carbonara
- guest_isAtRiskForFood meal_tagliatelle-alla-marinara

Data property assertions
- thing_hasName "Valentino Rossi"
- guest_hasLevelOfCalorieConscious 2
- guest_hasLevelOfCalorieConscious 3

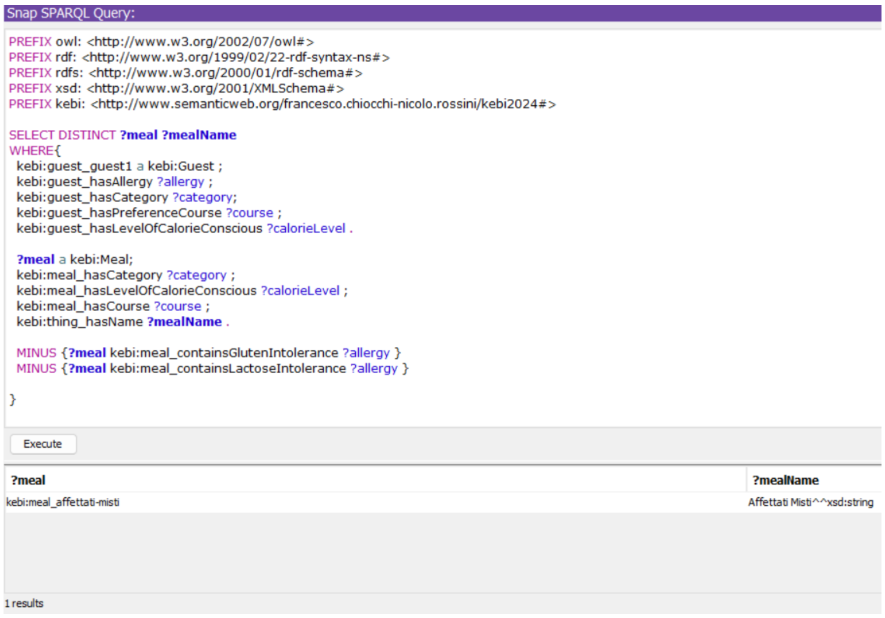
Negative object property assertions

Negative data property assertions

Figure 3.29: Guest1 Properties

```
SELECT DISTINCT ?meal ?mealName
WHERE{
  kebi:guest_guest1 a kebi:Guest ;
  kebi:guest_hasAllergy ?allergy ;
  kebi:guest_hasCategory ?category;
  kebi:guest_hasPreferenceCourse ?course ;
  kebi:guest_hasLevelOfCalorieConscious ?calorieLevel .

  ?meal a kebi:Meal;
  kebi:food_hasCategory ?category ;
  kebi:meal_hasLevelOfCalorieConscious ?calorieLevel ;
  kebi:meal_hasCourse ?course ;
  kebi:thing_hasName ?mealName .

  MINUS {?meal kebi:meal_containsGlutenIntolerance ?allergy }
  MINUS {?meal kebi:meal_containsLactoseIntolerance ?allergy }

}
```
Listing 3.15: Advanced query for see the meals in base of the preference of the guest

Results:

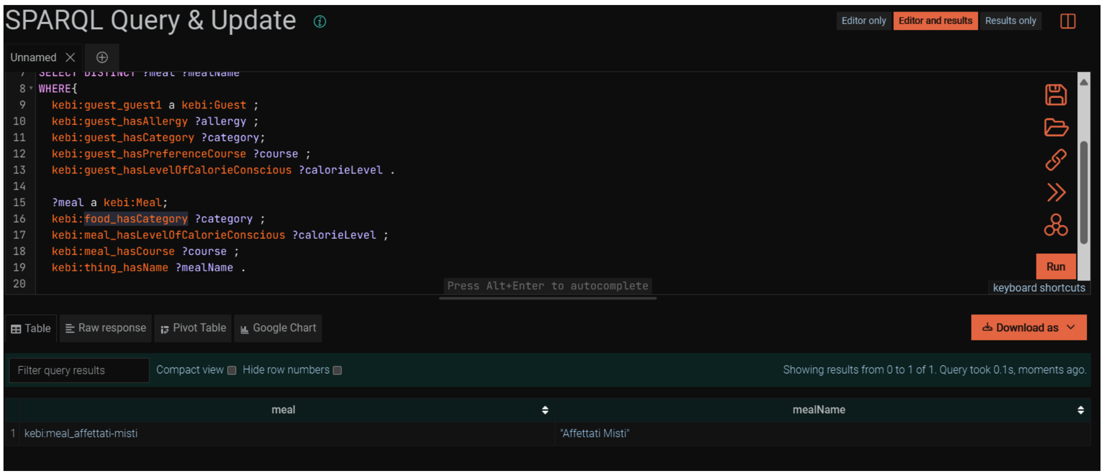


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#>

SELECT DISTINCT ?meal ?mealName
WHERE{
 kebi:guest_guest1 a kebi:Guest ;
 kebi:guest_hasAllergy ?allergy ;
 kebi:guest_hasCategory ?category;
 kebi:guest_hasPreferenceCourse ?course ;
 kebi:guest_hasLevelOfCalorieConscious ?calorieLevel .

 ?meal a kebi:Meal;
 kebi:meal_hasCategory ?category ;
 kebi:meal_hasLevelOfCalorieConscious ?calorieLevel ;
 kebi:meal_hasCourse ?course ;
 kebi:thing_hasName ?mealName .

 MINUS {?meal kebi:meal_containsGlutenIntolerance ?allergy }
 MINUS {?meal kebi:meal_containsLactoseIntolerance ?allergy }

}
```

?meal	?mealName
kebi:meal_affettati-misti	Affettati Misti^^xsd:string

1 results

Figure 3.30: SPARQL Query with Protégé: Guest1 Suggested Meals



```
SELECT DISTINCT ?meal ?mealName
WHERE{
  kebi:guest_guest1 a kebi:Guest ;
  kebi:guest_hasAllergy ?allergy ;
  kebi:guest_hasCategory ?category;
  kebi:guest_hasPreferenceCourse ?course ;
  kebi:guest_hasLevelOfCalorieConscious ?calorieLevel .

  ?meal a kebi:Meal;
  kebi:food_hasCategory ?category ;
  kebi:meal_hasLevelOfCalorieConscious ?calorieLevel ;
  kebi:meal_hasCourse ?course ;
  kebi:thing_hasName ?mealName .
```

Showing results from 0 to 1 of 1. Query took 0.1s, moments ago.

	meal	mealName
1	kebi:meal_affettati-misti	"Affettati Misti"

Figure 3.31: SPARQL Query with GraphDB: Guest1 Suggested Meals

The remaining queries will be listed but their results will not be provided due to the large number of queries.

- **Query for return Vegetarian meals**

```
SELECT ?meal
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:food_hasCategory kebi:category_vegetarian .
}
```

Listing 3.16: Vegetarian Meals

- **Query for view the meals and the related calories**

```
SELECT ?meal ?calories
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:food_hasKcal ?calories .
}
```

Listing 3.17: Meals and related calories

- **Query for view the guests with allergies**

```
SELECT ?guest ?allergy
WHERE {
  ?guest a kebi:Guest ;
         kebi:guest_hasAllergy ?allergy .
}
```

Listing 3.18: Guests with allergies

- **Query for view the kcal of all ingredients**

```
SELECT ?ingredient ?kcal
WHERE {
  ?ingredient a kebi:Ingredient .
  ?ingredient kebi:food_hasKcal ?kcal .
}
```

Listing 3.19: Kcal of all ingredients

- **Query for view the level 3 of Calorie Conscious of all meals**

```
SELECT DISTINCT ?meal ?calorie
WHERE {
  ?meal rdf:type ?type .
  ?meal kebi:meal_hasLevelOfCalorieConscious ?calorie .
  FILTER(?calorie = 3)
}
```

Listing 3.20: Meals with level 3 of Calorie Conscious

- **Query for view the kcal of a specific ingredient**

```
SELECT ?kcal
WHERE {
  kebi:ingredient_apple kebi:food_hasKcal ?kcal .
}
```

Listing 3.21: Kcal for a specific ingredient

- **Query for view the ingredients with has gluten intolerance**

```
SELECT ?ingredient
WHERE {
  ?ingredient rdf:type kebi:Ingredient .
  ?ingredient kebi:grain_containsGlutenIntolerance kebi:allergy_gluten
    .
}
```

Listing 3.22: Ingredients with gluten intolerance

- **Query for view the ingredients with has lactose intolerance**

```
SELECT ?ingredient
WHERE {
  ?ingredient rdf:type kebi:Ingredient .
  ?ingredient kebi:dairy_containsLactoseIntolerance kebi:
    allergy_lactose .
}
```

Listing 3.23: Ingredients with lactose intolerance

- **Query for view the meals with has gluten intolerance**: only with PROTEGE -Reasoner

```
SELECT ?meal
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:meal_containsGlutenIntolerance kebi:allergy_gluten .
}
```

Listing 3.24: Meals with gluten intolerance - REASONER Protégé

- **Query for view the meals with has lactose intolerance**: only with PROTEGE -Reasoner

```
SELECT ?meal
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:meal_containsLactoseIntolerance kebi:allergy_lactose .
}
```

Listing 3.25: Meals with lactose intolerance - REASONER Protégé

- **Query for view the meals that are classified as Appetizer**

```
SELECT ?meal
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:meal_hasCourse kebi:course_appetizer .
}
```

Listing 3.26: Meals classified as Appetizer

- **Query for view the meals that are classified as First Dish**

```
SELECT ?meal
WHERE {
  ?meal rdf:type kebi:Meal .
  ?meal kebi:meal_hasCourse kebi:course_first-dish .
}
```

Listing 3.27: Meals classified as First Dish

- **Query for view the ingredient that are classified as Seafood**

```
SELECT ?ingredient
WHERE {
  ?ingredient rdf:type kebi:Seafood.
}
```

Listing 3.28: Ingredient that are classified as Seafood

- **Query for view the meals that has more than 500 kcal**

```
SELECT ?meal ?kcal
WHERE {
  ?meal kebi:food_hasKcal ?kcal.
  FILTER(xsd:decimal(?kcal) > 500).
}
```

Listing 3.29: Meals that has more than 500 kcal

3.3.4 SHACL Shape

SHACL (Shapes Constraint Language) is a language used for validating RDF (Resource Description Framework) data against a set of conditions, known as shapes. These shapes define the expected structure and content of RDF graphs, ensuring data consistency and integrity within semantic web applications. SHACL shapes can specify constraints on the types of nodes, the cardinality of properties, and even the values that properties can take. By defining these constraints, SHACL helps maintain high data quality and enables more robust data integration and querying.

The SHACL Shape File can be found in the 'ontology' folder and is named 'shacl_menu'[8]. Here is a description of the file:

- **RestaurantShape**: Ensures that each Restaurant instance has at least one Menu (MenuShape). We did not set a maxCount of 1, allowing the restaurant manager the flexibility to add additional types of menus in the future. Additionally, it optionally allows associations with guests (GuestShape), who are added after scanning the QR code

[8]SHACL Shape File: https://github.com/FrancescoChiocchi8/KEBI-Project/blob/main/Task1/ontology/shacl_menu

```
kebi2024:RestaurantShape a sh:NodeShape ;
    sh:targetClass kebi2024:Restaurant ;
    sh:property [
        sh:path kebi2024:restaurant_hasMenu ;
        sh:node kebi2024:MenuShape ;
        sh:minCount 1 ;
        sh:message "It is impossible that a Restaurant does not have a
    menu!!" ;
    ] ;
    sh:property [
        sh:path kebi2024:restaurant_hasGuest ;
        sh:node kebi2024:GuestShape ;
        sh:minCount 0;
    ] ;
.
```

Listing 3.30: SHACL Shape for Restaurant

”If we delete the menu, for example, the following error will be returned: ”It is impossible that a Restaurant does not have a menu”.

- **MenuShape**: Defines constraints for instances of Menu, ensuring that it contains at least one Meal and one Drink.

```
kebi2024:MenuShape a sh:NodeShape ;
    sh:targetClass kebi2024:Menu ;
    sh:property [
        sh:path kebi2024:menu_containsMeal ;
        sh:node kebi2024:MealShape ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path kebi2024:menu_containsDrink ;
        sh:node kebi2024:DrinkShape ;
        sh:minCount 1 ;
    ] ;
.
```

Listing 3.31: SHACL Shape for Menu

Similarly, if the menu does not include any food or drinks, an error will occur.

- **GuestShape**: Validates instances of Guest, requiring mandatory information such as dietary category (CategoryShape) and calorie-consciousness level, while specifying optional details like allergies (AllergyShape). The calorie-consciousness level ranges from 1 to 4 because if a guest has a level of 0, it automatically includes levels 1, 2, and 3. Unfortunately, we were unable to use SWRL due to its inference limitations when running the Reasoner, which functions only within Protégé. For instance, on platforms like graphDB or AOAME, some data may be lost.

```
kebi2024:GuestShape a sh:NodeShape ;
    sh:targetClass kebi2024:Guest ;
    sh:property [
        sh:path kebi2024:guest_hasAllergy ;
        sh:node kebi2024:AllergyShape ;
        sh:minCount 0 ;
    ] ;
    sh:property [
        sh:path kebi2024:guest_hasCategory ;
        sh:node kebi2024:CategoryShape ;
        sh:minCount 1;
    ] ;
    sh:property [
        sh:path kebi2024:guest_isAtRiskForFood ;
        sh:node kebi2024:MealShape ;
    ] ;
    sh:property [
        sh:path kebi2024:guest_hasLevelOfCalorieConscious ;
        sh:datatype xsd:integer ;
        sh:minCount 1 ;
        sh:maxCount 4 ;
    ] ;
.
```

Listing 3.32: SHACL Shape for Guest

It has been considered that a guest may optionally have an allergy. In our case, we have only defined two types of allergies (but, for example, we could also consider nut allergies or similar). For the sake of extensibility, as previously done, a maxCount has not been set, allowing for the accommodation of new allergies that may emerge in the future.

- **MealShape**: Specifies constraints for instances of Meal, including optional intolerance to lactose (LactoseShape) and gluten (GlutenShape), categories, courses, and ingredients.

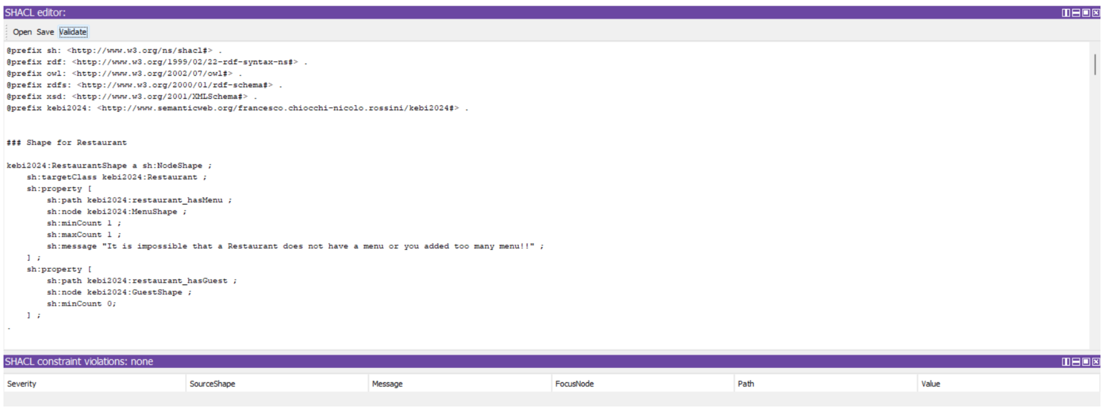
```
kebi2024:MealShape a sh:NodeShape ;
    sh:targetClass kebi2024:Meal ;
    sh:property [
        sh:path kebi2024:meal_containsLactoseIntolerance ;
        sh:node kebi2024:LactoseShape ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path kebi2024:meal_containsGlutenIntolerance ;
        sh:node kebi2024:GlutenShape ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path kebi2024:meal_hasCategory ;
        sh:node kebi2024:CategoryShape ;
        sh:minCount 0 ;
    ] ;
    sh:property [
        sh:path kebi2024:meal_hasCourse ;
        sh:node kebi2024:CourseShape ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path kebi2024:meal_hasIngredient ;
        sh:node kebi2024:IngredientShape ;
        sh:minCount 1 ;
    ] ;
.
```

Listing 3.33: SHACL Shape for Meal

In this case, a Meal can contain lactose or gluten, and it can have one or more categories. For example, a meal item might be suitable for both vegetarian and non-vegetarian guests. Additionally, if we consider vegans and the meal item is suitable for them as well, it will belong to that category too. Therefore, no maxCount has been set. Furthermore, a meal must have a course. Finally, to make a meal, there must be at least one ingredient.

Validation

Testing the aforementioned file yielded the following result:



```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix kebi2024: <http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#> .


### Shape for Restaurant

kebi2024:RestaurantShape a sh:NodeShape ;
    sh:targetClass kebi2024:Restaurant ;
    sh:property [
        sh:path kebi2024:restaurant_hasMenu ;
        sh:node kebi2024:MenuShape ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:message "It is impossible that a Restaurant does not have a menu or you added too many menu!!" ;
    ] ;
    sh:property [
        sh:path kebi2024:restaurant_hasGuest ;
        sh:node kebi2024:GuestShape ;
        sh:minCount 0;
    ] ;
.
```

SHACL constraint violations: none

Severity	SourceShape	Message	FocusNode	Path	Value

Figure 3.32: SHACL Validator

As we can see at the end of image, no violations were found.

4. Agile and Ontology-based Meta-Modelling

In Section 4.1, we will describe what AOAME is, while in Section 4.2, we will outline what BPMN 2.0 entails and present our model.

Here you can find the link to our project for Task 2 [1].

4.1 AOAME

AOAME [2] is a prototypical tool designed to implement an agile, ontology-based approach to meta-modeling. This approach aims to design and maintain schemas for Enterprise Knowledge Graphs (EKGs), which structure the knowledge of a specific application domain, enabling analysis, reasoning, and integration of information extracted from various data sources. The primary challenge in developing and maintaining an EKG schema is the requirement for expertise in both ontology engineering and the application domain.
The proposed approach extends traditional meta-modeling with agile principles, allowing for domain-specific adaptations of modeling languages and real-time testing. This integration facilitates the involvement of domain experts in the engineering cycle. AOAME, developed following the Design Science Research methodology, serves as a prototype to evaluate the utility of this approach. It provides capabilities to extend, modify, and remove modeling constructs, and to hide them from the tool palette. These actions are supported by meta-modeling operators that generate SPARQL statements and update the triplestore, ensuring consistency between human-interpretable and machine-interpretable knowledge.
The software architecture of AOAME leverages Java libraries of Fuseki to develop APIs that retrieve ontologies from the triplestore and display them in a GUI, as well as to implement the meta-modeling operators. This tool has been validated through the implementation of real-world scenarios and significant case studies, demonstrating its operability and generality across various application domains. AOAME facilitates close cooperation between domain experts and language engineers during the engineering of domain-specific modeling languages (DSMLs), thus enhancing the adaptability and effectiveness of the meta-modeling process.

[1]GitHub Forked Project link:
https://github.com/FrancescoChiocchi8/Ontology4ModelingEnvironment

[2]AOAME Reference: https://emisa-journal.org/emisa/article/view/310

4.2 BPMN 2.0

BPMN stands for Business Process Model and Notation. It is a graphical representation used to specify business processes in a workflow. BPMN provides a standardized way to visualize the sequence of business activities and the flow of information between them. It is designed to be understandable by all business stakeholders, including business analysts, process participants, and technical developers.

The key elements of BPMN 2.0 are:

- Events: Represent occurrences that affect the flow of the process (e.g., start events, end events, intermediate events).
- Activities: Tasks or work that needs to be performed (e.g., user tasks, service tasks, manual tasks).
- Gateways: Decision points that control the divergence and convergence of the process flow (e.g., exclusive gateways, parallel gateways, inclusive gateways).
- Pools and Lanes: Represent major participants in the process, with pools representing different organizations or entities, and lanes representing subdivisions within those entities.
- Artifacts: Supplementary elements such as data objects, groups, and annotations that provide additional information about the process.

In this project, we have decided to utilize BPMN to assist the restaurant manager in understanding the customer ordering process. Specifically, the BPMN diagram will consider both allergies and dietary preferences. Customers will be required to declare any allergies and indicate if they are vegetarian, vegan, or carnivorous. Subsequently, they can decide on the types of meals they would like to order. This approach ensures that the ordering process is tailored to individual needs, enhancing customer satisfaction and safety.

4.2.1 Our BPMN 2.0 Model

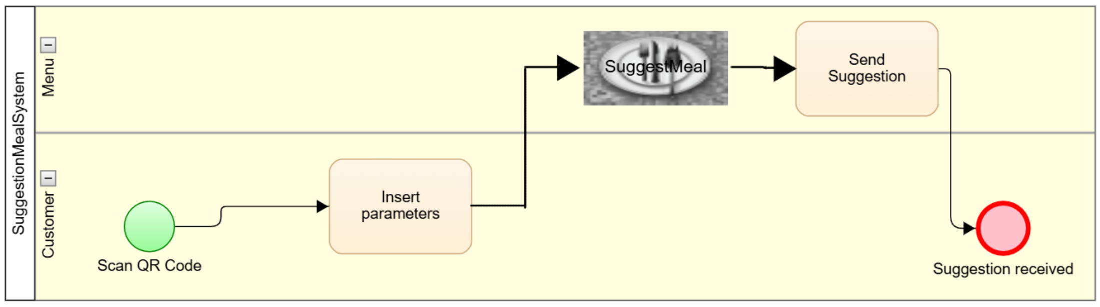
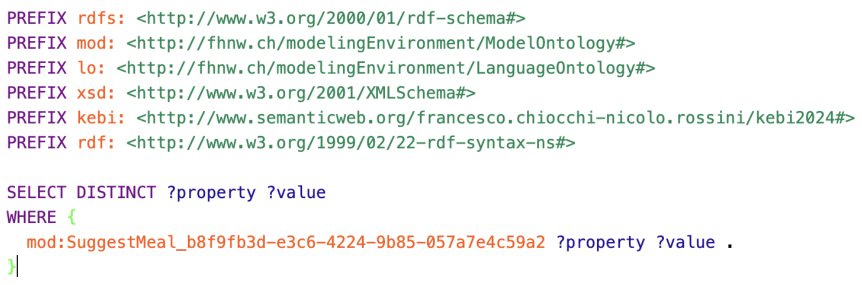


Figure 4.1: BPMN 2.0 graph built in AOAME

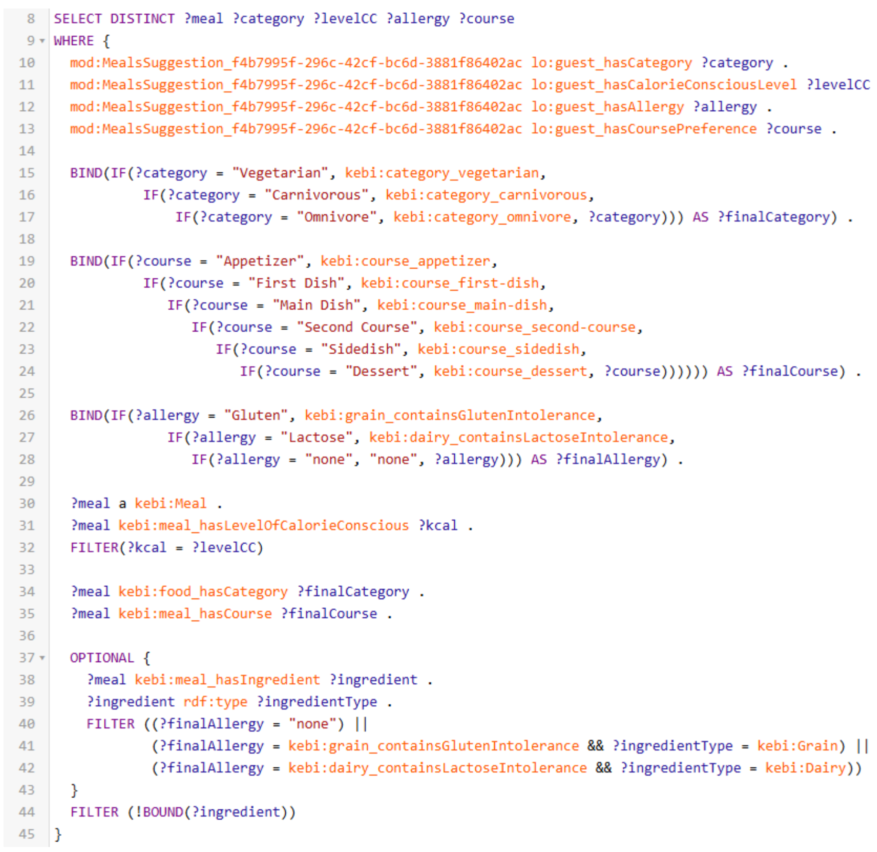
As we can see, we have inserted some particular elements to complete the graph. In particular we used a pool element to specify the menu and customer that interact during the whole process. In other words, the customer scan the QR code to see the menu, than he must insert the parameter like allergies and food category, finally the tool provides the suggestions about meals. By implementing this task, we had the opportunity to work with AOAME (Agile and Ontology-Aided Modeling Environment) which is a methodology and toolset designed to enhance software development and business process modeling by combining agile principles with ontology-based techniques.

After building the BPMN 2.0 we used Jena Fuseki to create some query for examining the graph.
Apache Jena Fuseki is a robust and scalable SPARQL server designed to handle RDF data efficiently. It provides a RESTful interface to perform various operations on RDF datasets, including: Execute SPARQL queries to retrieve data; Perform SPARQL updates to modify data; Load, store, and manage RDF datasets.



```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mod: <http://fhnw.ch/modelingEnvironment/ModelOntology#>
PREFIX lo: <http://fhnw.ch/modelingEnvironment/LanguageOntology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?property ?value
WHERE {
  mod:SuggestMeal_b8f9fb3d-e3c6-4224-9b85-057a7e4c59a2 ?property ?value .
}
```

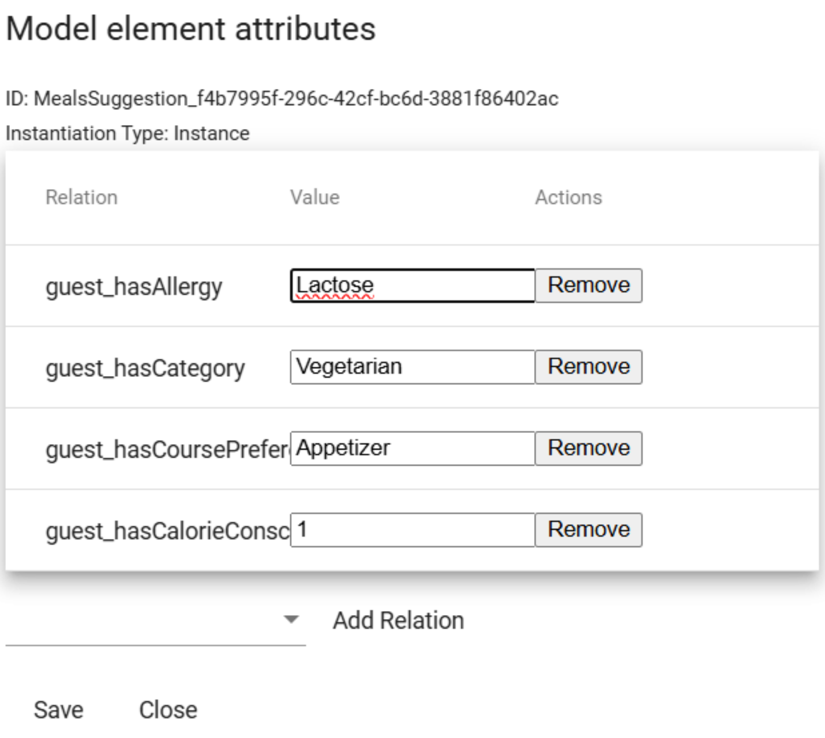
Figure 4.2: Query 1: used for see the property and their relative value of our extended class SuggestMeal.



```
SELECT DISTINCT ?meal ?category ?levelCC ?allergy ?course
WHERE {
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCategory ?category .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCalorieConsciousLevel ?levelCC
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasAllergy ?allergy .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCoursePreference ?course .

  BIND(IF(?category = "Vegetarian", kebi:category_vegetarian,
          IF(?category = "Carnivorous", kebi:category_carnivorous,
              IF(?category = "Omnivore", kebi:category_omnivore, ?category))) AS ?finalCategory) .

  BIND(IF(?course = "Appetizer", kebi:course_appetizer,
          IF(?course = "First Dish", kebi:course_first-dish,
             IF(?course = "Main Dish", kebi:course_main-dish,
                IF(?course = "Second Course", kebi:course_second-course,
                   IF(?course = "Sidedish", kebi:course_sidedish,
                      IF(?course = "Dessert", kebi:course_dessert, ?course)))))) AS ?finalCourse) .

  BIND(IF(?allergy = "Gluten", kebi:grain_containsGlutenIntolerance,
             IF(?allergy = "Lactose", kebi:dairy_containsLactoseIntolerance,
                IF(?allergy = "none", "none", ?allergy))) AS ?finalAllergy) .

  ?meal a kebi:Meal .
  ?meal kebi:meal_hasLevelOfCalorieConscious ?kcal .
  FILTER(?kcal = ?levelCC)

  ?meal kebi:food_hasCategory ?finalCategory .
  ?meal kebi:meal_hasCourse ?finalCourse .

  OPTIONAL {
    ?meal kebi:meal_hasIngredient ?ingredient .
    ?ingredient rdf:type ?ingredientType .
    FILTER ((?finalAllergy = "none") ||
            (?finalAllergy = kebi:grain_containsGlutenIntolerance && ?ingredientType = kebi:Grain) ||
            (?finalAllergy = kebi:dairy_containsLactoseIntolerance && ?ingredientType = kebi:Dairy))
  }
  FILTER (!BOUND(?ingredient))
}
```

Figure 4.3: Query 2: used for analyze the extended class and return the meals which satisfied all data property described.

The first query retrieves the properties and their relative value of SuggestMeal class, while the second one is used to analyze the graph and it provides the meal suggestions that satisfy the customer's parameters.



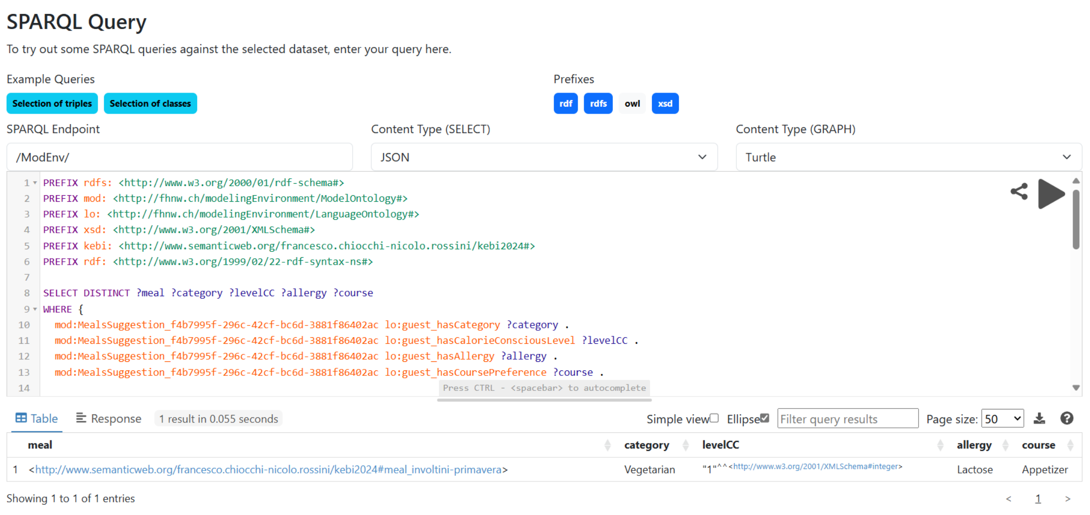
Model element attributes

ID: MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac

Instantiation Type: Instance

Relation	Value	Actions
guest_hasAllergy	Lactose	Remove
guest_hasCategory	Vegetarian	Remove
guest_hasCoursePrefer	Appetizer	Remove
guest_hasCalorieConsc	1	Remove

Figure 4.4: Define the attributes of our extended class. We set up the lactose allergy with vegetarian category.



SPARQL Query

To try out some SPARQL queries against the selected dataset, enter your query here.

SPARQL Endpoint: /ModEnv/

Content Type (SELECT): JSON

Content Type (GRAPH): Turtle

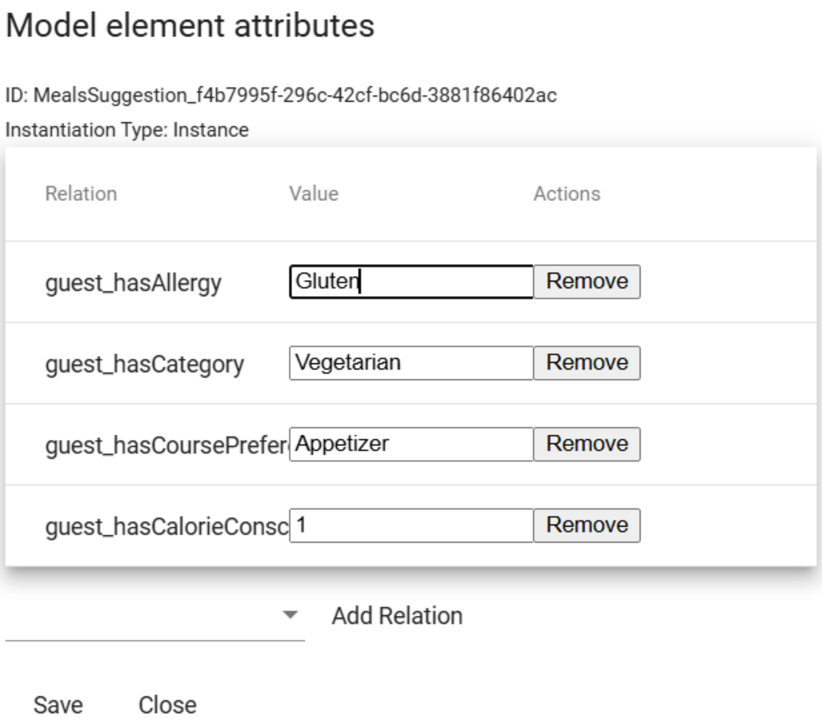
```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mod: <http://fhnw.ch/modelingEnvironment/ModelOntology#>
PREFIX lo: <http://fhnw.ch/modelingEnvironment/LanguageOntology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?meal ?category ?levelCC ?allergy ?course
WHERE {
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCategory ?category .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCalorieConsciousLevel ?levelCC .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasAllergy ?allergy .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCoursePreference ?course .
                                                             Press CTRL - <spacebar> to autocomplete
```

	meal	category	levelCC	allergy	course
1	<http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#meal_involtini-primavera>	Vegetarian	"1"^^<http://www.w3.org/2001/XMLSchema#integer>	Lactose	Appetizer

Showing 1 to 1 of 1 entries

Figure 4.5: Result of query 2 with parameter of figure 4.4

For the first example, as we can see in the image above, we set up the attributes of our BPMN class and we fire the query in Jena Fuseki that gives back the result.



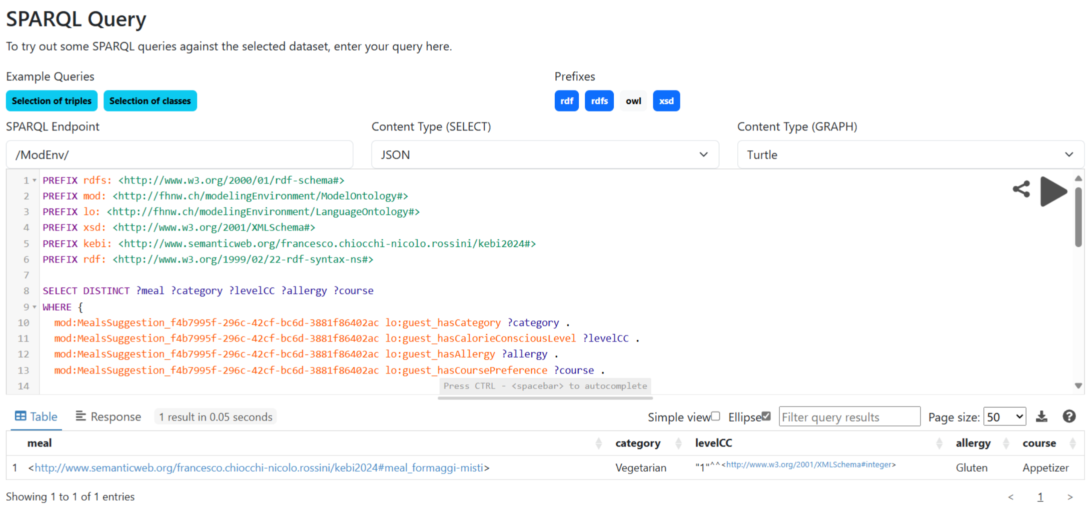
Model element attributes

ID: MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac

Instantiation Type: Instance

Relation	Value	Actions
guest_hasAllergy	Gluten	Remove
guest_hasCategory	Vegetarian	Remove
guest_hasCoursePrefer	Appetizer	Remove
guest_hasCalorieConsc	1	Remove

Figure 4.6: Define the attributes of our extended class. We set up the gluten allergy with vegetarian category.



SPARQL Query

To try out some SPARQL queries against the selected dataset, enter your query here.

Example Queries

Prefixes

SPARQL Endpoint: /ModEnv/

Content Type (SELECT): JSON

Content Type (GRAPH): Turtle

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mod: <http://fhnw.ch/modelingEnvironment/ModelOntology#>
PREFIX lo: <http://fhnw.ch/modelingEnvironment/LanguageOntology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?meal ?category ?levelCC ?allergy ?course
WHERE {
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCategory ?category .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCalorieConsciousLevel ?levelCC .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasAllergy ?allergy .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCoursePreference ?course .
                                                  Press CTRL - <spacebar> to autocomplete
```

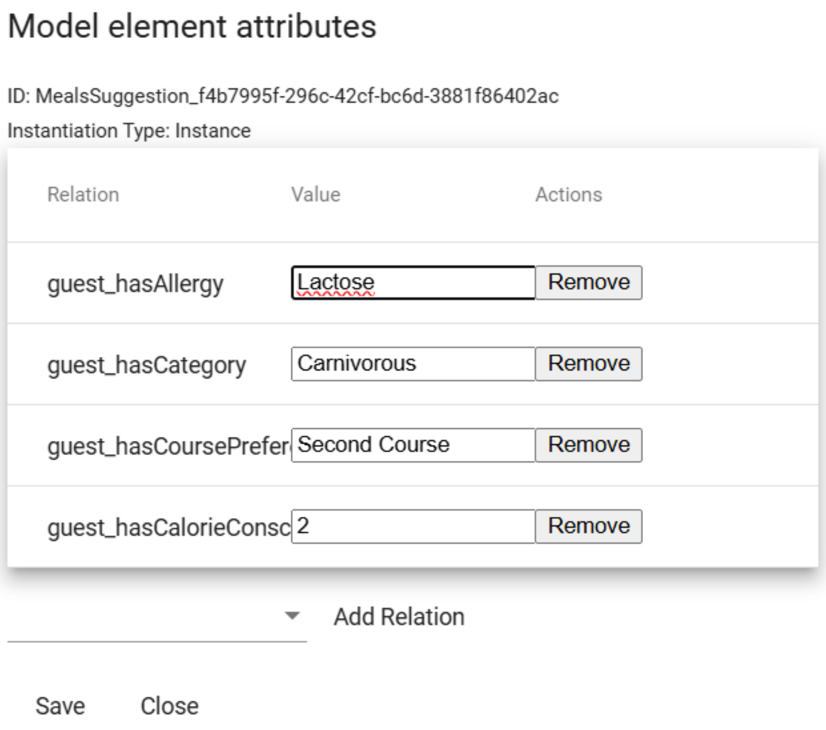
1 result in 0.05 seconds

Simple view☐ Ellipse☑ Filter query results Page size: 50

	meal	category	levelCC	allergy	course
1	<http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#meal_formaggi-misti>	Vegetarian	"1"^^<http://www.w3.org/2001/XMLSchema#integer>	Gluten	Appetizer

Showing 1 to 1 of 1 entries

Figure 4.7: Result of query 2 with parameter of figure 4.6



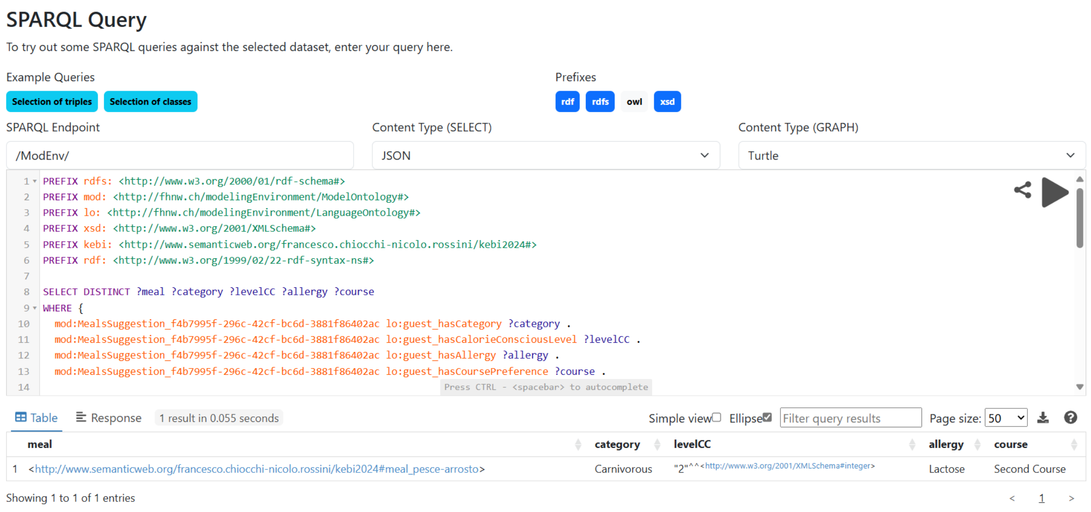
Model element attributes

ID: MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac

Instantiation Type: Instance

Relation	Value	Actions
guest_hasAllergy	Lactose	Remove
guest_hasCategory	Carnivorous	Remove
guest_hasCoursePrefer	Second Course	Remove
guest_hasCalorieConsc	2	Remove

Figure 4.8: Define the attributes of our extended class. We set up the lactose allergy with carnivorous category.



SPARQL Query

To try out some SPARQL queries against the selected dataset, enter your query here.

SPARQL Endpoint: /ModEnv/

Content Type (SELECT): JSON

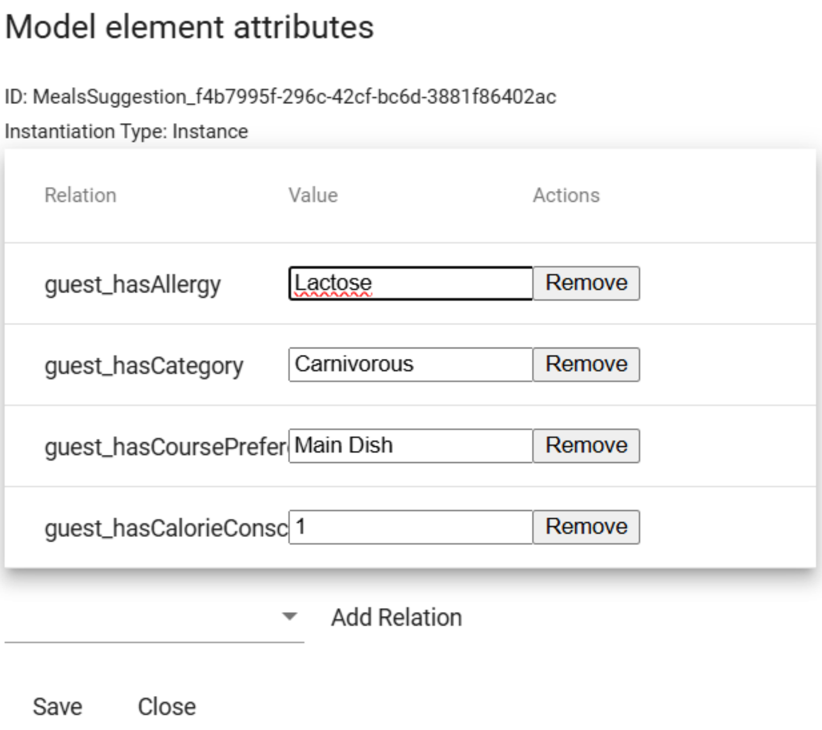
Content Type (GRAPH): Turtle

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mod: <http://fhnw.ch/modelingEnvironment/ModelOntology#>
PREFIX lo: <http://fhnw.ch/modelingEnvironment/LanguageOntology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?meal ?category ?levelCC ?allergy ?course
WHERE {
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCategory ?category .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCalorieConsciousLevel ?levelCC .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasAllergy ?allergy .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCoursePreference ?course .
                                                  Press CTRL - <spacebar> to autocomplete
```

	meal	category	levelCC	allergy	course
1	<http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#meal_pesce-arrosto>	Carnivorous	"2"^^<http://www.w3.org/2001/XMLSchema#integer>	Lactose	Second Course

Showing 1 to 1 of 1 entries

Figure 4.9: Result of query 2 with parameter of figure 4.8



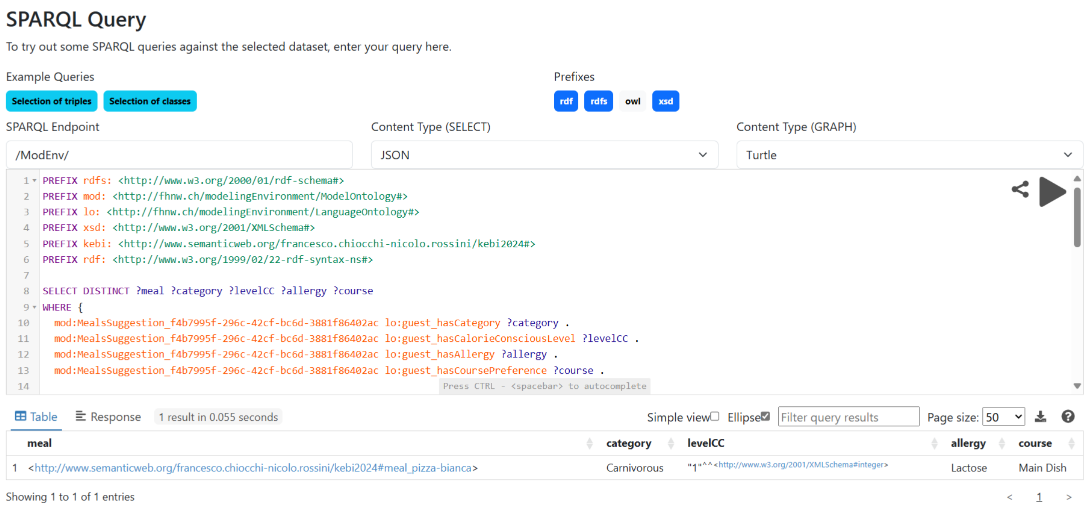
Model element attributes

ID: MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac

Instantiation Type: Instance

Relation	Value	Actions
guest_hasAllergy	Lactose	Remove
guest_hasCategory	Carnivorous	Remove
guest_hasCoursePrefer	Main Dish	Remove
guest_hasCalorieConsc	1	Remove

Figure 4.10: Define the attributes of our extended class. We set up the lactose allergy with carnivorous category and main dish.



SPARQL Query

To try out some SPARQL queries against the selected dataset, enter your query here.

SPARQL Endpoint: /ModEnv/

Content Type (SELECT): JSON

Content Type (GRAPH): Turtle

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mod: <http://fhnw.ch/modelingEnvironment/ModelOntology#>
PREFIX lo: <http://fhnw.ch/modelingEnvironment/LanguageOntology#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?meal ?category ?levelCC ?allergy ?course
WHERE {
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCategory ?category .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCalorieConsciousLevel ?levelCC .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasAllergy ?allergy .
  mod:MealsSuggestion_f4b7995f-296c-42cf-bc6d-3881f86402ac lo:guest_hasCoursePreference ?course .
                                                  Press CTRL - <spacebar> to autocomplete
```

1 result in 0.055 seconds

Simple view ☐ Ellipse ☑ Filter query results Page size: 50

	meal	category	levelCC	allergy	course
1	<http://www.semanticweb.org/francesco.chiocchi-nicolo.rossini/kebi2024#meal_pizza-bianca>	Carnivorous	"1"^^<http://www.w3.org/2001/XMLSchema#integer>	Lactose	Main Dish

Showing 1 to 1 of 1 entries

< 1 >

Figure 4.11: Result of query 2 with parameter of figure 4.10

5. Conclusions

5.1 Francesco Chiocchi

During this project, my colleague and I engaged extensively in knowledge engineering, exploring various knowledge-based solutions, each with its own set of advantages and disadvantages. I will now provide an overview of my impressions on these solutions, highlighting both positive and negative aspects.

- **Decision Tables**: Decision tables operate straightforwardly, taking inputs and generating outputs based on predefined rules organized in rows. The selection of outputs follows a "Hit Policy," which dictates how results are grouped. Our project employed a "Collect" policy, enabling concurrent evaluation of all rules, maximizing the potential of Decision Model and Notation (DMN). Under this policy, input conditions are structured in a simple format where columns are linked by an "and" relationship, while rows operate under an "or" relationship with each other. Output results are aggregated into sets for respective output columns. This operational clarity makes it user-friendly and straightforward to implement. However, in complex contexts with numerous variables and intricate conditions, the applicability of decision tables may become impractical or exceedingly intricate. Manipulating data using decision tables also proved to be cumbersome and repetitive. The manual creation of entries became monotonous and tedious, compounded by difficulties in debugging, exacerbated by limited support from available online tools.

- **Prolog**: I thoroughly enjoyed working with Prolog during this project. It provided me with an opportunity to explore a new programming language and conduct extensive experimentation. Prolog, although not excessively challenging to grasp, presents difficulties in accessing comprehensive learning resources. It emphasizes recursion heavily, lacks conventional loop structures, and straightforward arithmetic operations. However, it excels in scenarios where managing data and retrieving information from knowledge bases are straightforward and intuitive. Despite the enjoyable learning experience, Prolog has significant drawbacks. Chief among these is the lack of clarity in some areas. Additionally, while Prolog is optimized for logical operations, handling large knowledge bases can strain its interpreter, resulting in extended computation times for obtaining results.

- **Protégé**: I viewed this approach as a hybrid between DMN (Decision Model and Notation) and Object-Oriented programming languages. In this framework, we could define objects along with their relationships and attributes, and extract knowledge from these object instances. This method of modeling knowledge intrigued me, particularly due to its querying method closely resembling standard SQL, which simplified its usage. I found the SHACL validator highly useful for testing whether a property is correctly defined or not, likening it to performing assertTrue or assertFalse in Java. SWRL rules enable type inference, but in my view, they should be used sparingly and judiciously. This perspective arises from experiences with querying tools like AOAME or GraphDB, which lack built-in reasoning capabilities for type inference. As a result, queries needed to be meticulously crafted, especially due to rules defined for meals, where inference determined whether a meal contained allergenic ingredients based on its constituents.

- **AOAME**: From my perspective, I found AOAME to be an extremely valuable tool due to its ability to adapt BPMN 2.0 with our ontology. This allowed us to create a diagram that is much simpler and easier to read compared to many others that do not utilize this tool. Specifically, by creating the class "Suggest Meal", a subclass of "Task", we were able to develop a new graphical notation that is very intuitive for the restaurant manager. Additionally, we introduced four new types of properties (course, allergy, calorie-conscious, and category) to integrate them into our ontology, effectively combining them.
  By employing Apache Jena Fuseki, we seamlessly integrated our ontology into a triplestore. This integration enabled us to execute SPARQL queries, dynamically generating personalized menu suggestions that cater to each guest's unique needs. The process of creating queries to suggest meals based on user preferences was straightforward, largely due to the similarity in query development with Protégé. However, AOAME faces challenges such as significant bugs that impact user experience and system stability. Additionally, its online version may struggle during periods of high user traffic, potentially limiting operational efficiency.
  Despite these cons, AOAME remains a very powerful and highly useful tool.

In conclusion, each knowledge-based solution has its strengths and weaknesses. Decision tables are straightforward and user-friendly but can become cumbersome in complex scenarios. Prolog offers intuitive data management but struggles with large knowledge bases and lacks comprehensive learning resources. Protégé provides a powerful hybrid approach for object-oriented knowledge modeling but requires careful rule management. AOAME excels in integrating meta-modeling and ontologies for enhanced data integration and reasoning but suffers from stability issues and high user traffic impacts. Choosing the appropriate solution depends on the specific needs of the project, the complexity of the knowledge base, and the desired level of customization and scalability.

5.2 Nicolò Rossini

The project aimed to enhance the dining experience by creating a system that customizes restaurant menus according to guest preferences and dietary restrictions. This initiative addresses the challenge of navigating large digital menus on small screens and ensures guests are presented with only the meals they can or prefer to consume.
We developed multiple knowledge-based solutions to recommend meals based on guest profiles using various representation languages:

- **Decision Tables**:Working with decision tables during this project highlighted their practicality and simplicity in handling rule-based logic. Decision tables offer a structured approach, where inputs are processed to generate outputs based

on a set of predefined rules arranged in rows.
Despite the user-friendly nature of decision tables, they are not without their limitations. In more complex scenarios, where the number of variables and conditions increases significantly, decision tables can become difficult to manage. The simplicity that makes them appealing for straightforward tasks can turn into a hindrance when dealing with intricate conditions, leading to impractical or overly complicated implementations. Additionally, the process of manually creating and manipulating data entries can be monotonous and time-consuming. Debugging issues further exacerbates these challenges, especially given the limited support available from existing online tools.
In essence, while decision tables provide a clear and accessible way to manage rule-based logic, their effectiveness diminishes as complexity grows. They are best suited for relatively simple decision-making processes, where their structured approach can be fully utilized without being overwhelmed by the intricacies of more elaborate conditions and data sets.

- **Prolog**: Working with Prolog introduced me to a unique paradigm of programming, which sparked curiosity and a desire to experiment with new ways of thinking about code.
  Prolog shines in scenarios that involve logical reasoning and querying knowledge bases, where its declarative nature makes data management and retrieval both intuitive and powerful. However, these strengths are counterbalanced by some notable limitations. For instance, the clarity of Prolog's syntax and logic can sometimes be ambiguous, which may lead to confusion during development. Moreover, while Prolog is highly efficient for logical operations, it struggles with performance when handling large knowledge bases. This can result in slow processing times, which detracts from its overall efficiency in larger-scale applications.
  In summary, while Prolog offers a fascinating approach to problem-solving, particularly in logic-intensive tasks, its learning curve, ambiguity, and performance issues make it a less practical choice for projects that involve substantial data processing or require quick and clear-cut solutions.

- **Knowledge Graph/Ontology**: We created an ontology to represent the detailed knowledge about meals, ingredients, and guest preferences. Using Semantic Web Rule Language (SWRL) for rules and SPARQL for queries. SHACL shapes were employed to validate data consistency. This approach facilitated a rich and extensible framework for capturing and utilizing domain knowledge, allowing for flexible and dynamic interactions with the data.
  In addition, I want to add a brief description of these kind of tools: SWRL is a powerful language that extends the capabilities of OWL (Web Ontology Language) by enabling the creation of complex rules for inferring new knowledge within an ontology. For example, SWRL can infer whether a meal contains allergenic ingredients based on the ingredients' properties. This can significantly enhance the ontology's reasoning capabilities, allowing for the automatic classification of data and the discovery of implicit relationships. However, one of the disadvantages of SWRL is that it can introduce performance challenges, especially when dealing with large datasets or complex rules, as the reasoning process can become computationally intensive.
  On the other hand, SHACL is a language used to validate RDF (Resource Description Framework) graphs against a set of constraints or "shapes." It ensures

that the data in the ontology conforms to expected structures and values, which is critical for maintaining data integrity and preventing errors. The advantage of using SHACL lies in its ability to provide robust validation mechanisms, similar to running unit tests in traditional programming languages. This helps in catching inconsistencies early in the development process. However, one downside is that SHACL requires a careful and thorough definition of shapes, which can become time-consuming and complex, especially as the ontology grows in size and scope. In summary, by integrating SWRL for advanced rule-based reasoning and SHACL for rigorous data validation, our ontology framework became both powerful and flexible, capable of handling intricate domain knowledge about meals, ingredients, and guest preferences. The advantages of this approach include enhanced reasoning capabilities and strong data integrity, though these come at the cost of increased complexity and potential performance issues in large or highly detailed ontologies.

Building on our ontology, we adapted BPMN 2.0 to suggest meals tailored to individual guests. We created a graphical notation that is intuitive for restaurant managers. This notation helps visualize the decision process in selecting appropriate meals based on guest profiles. Using Apache Jena Fuseki, we integrated our ontology into a triple store and executed SPARQL queries to dynamically generate personalized menu suggestions.
The system we developed has significant potential to improve guest satisfaction by providing personalized dining experiences. It also streamlines the decision-making process for restaurant managers, ensuring that guests are only shown meals that meet their dietary needs and preferences.
In conclusion, our project successfully demonstrates the integration of advanced knowledge representation techniques with practical applications in the restaurant industry, paving the way for more intelligent and user-friendly digital menu systems.