

## **1. Goal**

*Our goal question: What are the best products for the user based on their skin type and personal preferences?*

Skincare is an essential part of many people's daily routine. Considering the major effect that it can have on the skin, it is important for one to use products that are suitable for their skin type to prevent irritation and damage. However, with the fast-growing popularity of the skincare industry and large amount of products released onto the market every year, it can be difficult to keep track of all the different skincare items and their purposes. The goal of our application, therefore, is to present the user with an overview of products that best fit their needs based on their input. In order to do so, the user is presented with a list of questions. The results of this questionnaire form the base of the application by retrieving related data from a set of ontologies and datasets. A clear overview of recommended skincare products for the user's skin will be the resulting output. It will inform the user about the products' properties which, for instance, include the type of product (moisturizers, cleansers, etc.), suitable skin type, brand, price, ranking, and ingredients. In addition to finding recommended products, the application also allows the user to retrieve more information about individual brands and product ingredients. This will help the user understand more of the products that are bought and the brands from which they buy them.

## **2. Stakeholders**

The analysis is intended for skincare companies and people with skin issues. These stakeholders both want the best skincare products. This application can help with their needs by retrieving information, such as information on specific ingredients, products, and price range. The application will work the same for both stakeholders.

Firstly, skincare companies as a stakeholder need to investigate and research the current trends in the market, which can depend on features, such as the price and ingredients of the products. As a result, this application will give companies an idea of products worth investing in and worth selling to their customers to maximize their profit. It is also a huge benefit for the companies because their customers will see their products when using the application as a recommended output. In other words, it can also be used as an indication to rate the product's position in the market. The application will constantly be updated, thus allowing companies to keep track of the newest skincare on the market.

The second stakeholders would be for people with skin issues. As for people with skin issues, they want to find products that are just right for their skin type. Since skincare products are constantly being improved, it is hard for consumers to stay up to date with the best skincare products. This application will provide these people with an idea on what products to buy depending on their preferences. It will give them recommendations depending on their skin type (oily, dry, normal) which will prepare a list of different brands of the product, the ingredient list, rankings of the product, pricing, and so on.


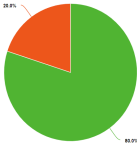


## **3. Design**

The aim of the design is to lay out the important features that have been selected for the customer, in order to display the essential information to the customer in an efficient manner. As a result, the visualization of the data has been decided to be displayed through a table containing the following features.

- a. appropriate skin type or disease for the product (example image)

- i. Extra information about the skin types can be found in the questionnaire that the user is presented with when starting the application. An overview will be given with the properties of different skin types and a small description, making it easier for the user to recognize their own.
- b. product name
- c. main ingredients of the product
- d. effects of the product
- e. side effects of the product
- f. manufacturer or brand of the product
- g. customer rating of the product
- h. price of the product in US dollars

The features above have been selected because they are asked and answered in the list of questions that the customer receives. In other words, the results will produce an ontology, which will contain the appropriate information to fill in the data for the questioned features. A table is capable of organizing and arranging complicated data by categorizing them into rows and columns. The arrangement of having certain types of information contained in a specific row or column will allow the customer to retrieve the information that is needed quickly and easily. A visual example of the table can be seen in Table 1. Different visual representations, such as a list would be less suitable, in comparison to a table, due to the fact that it would not be able to organize lengthy pieces of data. Moreover, lists do not contain images, thus when presenting data such as skin type will have to be presented in words. Therefore, the customer most likely will have to search up an example of the skin type to check whether it is the correct skin type that they have. Therefore, to effectively communicate the relevant information to the customers, in this situation a table would be the most appropriate choice.

| Product Name        | Skin type   | Ingredients          | Effects       | Side effects | Brand   | Rating  | Price in US dollars |
|---------------------|---|----------------------|---------------|--------------|---------|---|---------------------|
| tea tree cleanser   | <br>Greasy, acne | tea tree             | reduce acne   | dries skin   | l'oreal |  | 10.93               |
| coral reef cleanser | <br>Pores        | coral reef, sea salt | cleanse pores | burning      | garnier |  | 6.20                |

**Table 1:** Example Image of Data Layout Design

#### **4. Identify 2 ontologies**

The first ontology we want to use is called the Schema.org [1] ontology. Schema.org contains vocabulary that can be used in applications that publish, discover, and integrate different kinds of data. This ontology has different classes, which include products and services. It is especially useful to online stores with a great diversity of products. We can use this ontology to describe the relevant information about skincare products. By using this we can show the different products with prices, reviews, and quality of the product. Customers will also be able to find related products and see images of each skincare product. Using this ontology will make our application user-friendly because they do not have to search again to find similar products.

The second ontology we want to use is called Product vocabulary [2]. This ontology can be used to represent more information about the skincare products. By using this ontology, the customer will be able to search for products based on brand name, products and packages of products. By combining these two ontologies, it will be possible to make an application that can help users to find skincare products based on their preferences.

#### **5. External sources of data**

To create instances to use in the analysis, external sources of data have to be extracted. The first dataset, the cosmetics dataset [3], provides approximately 1,500 products including the product name, brand, rank, combination, ingredients, skin preference and price. This dataset will be capable of providing instances for the classes or subclasses within the ontology. In other words, it will further advance the quality of the recommendation system for the appropriate product and to satisfy the customer by being able to provide further information about the products, if needed.

DBpedia [4] is a database consisting of subsets of datasets in RDF format that offers various information when it comes to skincare such as pages on different brands [5], and on skincare as a subject itself [6]. All this information is useful to our application since it provides good additional information about product brands, but also ingredients which can be of great importance to the user (take for instance, people with allergies who want to know more about the ingredients). Because DBpedia has a SPARQL endpoint [7], it will enable the application to go through a wide range of brands and ingredients and make searching for instances based on the given query more efficient. In addition to its easily navigable database, DBpedia is a piece of open-source software used to gather and update the information about the products. This means it may be used and redistributed without requiring permission. For these reasons, the use of DBpedia is ideal as it can considerably improve the application's searching mechanism, and the quality and size of the output.

#### **6. Domain & scope of the ontology**

The explored domain is about skin products, in order to reduce the effort of searching for relevant information when having to purchase skin products from a customer's point of view. As consumers are being bombarded by advertisements and irrelevant information when searching for skin products, it becomes more obvious that there is an urgent need for an application capable of selecting products based on the customer's needs. The ontology will include data about the product, customer rating, and skin conditions. The application will need to retrieve the appropriate information, based on the question sheet or filter filled in by the user, which will then get processed to be displayed for the user to see. Therefore, the goal of this application is to build an ontology to help retrieve and display the correct output.

## 7. Methodology used in the construction of the ontology

Having a firm sight of what the main function of the application would be, it was decided that reusing two external ontologies and datasets would help create the ontology. The useful classes, subclasses, instances, relations, and properties, etc were selected from the external ontologies and datasets to build up the ontology as a base guideline. In addition, relationships and restrictions between the classes were created to add more accurate details when giving the correct output. Moreover, the data points about the product (instances) from the external sources would contribute to being able to provide more relevant information and actual outputs. Once the data has gone through the reasoner any errors or inconsistencies were taken care of to check whether the ontology was working properly. Possible questions that can be answered with the application would be: For which skin type is a product suited? Which type of products does a brand sell? Which skin care product has a rating higher than 4? The requirements to answer such questions is that we need specific data on products, product brands, ingredients and skincare product ratings.

## 8. Conceptualization

The domain of our ontology will cover skincare products and will consist of 21 classes. These 21 classes contain 5 main classes and 16 subclasses. The properties for our ontology consist of 8 different DataType properties and 6 different ObjectType properties. The relations between the classes, the DataType properties, and the ObjectType properties can be seen in a graph shown in *Appendix A, figure 1*.

### Classes:

- **ProductBrand** → A class of individual brands
- **ProductName** → A class of individual products that are linked to other instances using the properties available in the ontology
  - **ProductForSensitive** → A class of products for sensitive skin
  - **ProductForDry** → A class of products for dry skin
  - **ProductForNormal** → A class of products for normal skin
  - **ProductForOily** → A class of products for oily skin
  - **ProductForCombination** → A class of products for combination skin
- **ProductIngredient** → A class of individual ingredients
- **ProductType** → A class of product types
  - **Cleanser** → a product type that contains all the products that are cleansers
  - **Moisturizer** → a product type that contains all the products that are moisturizers
  - **Eye\_Cream** → a product type that contains all the products that are eye creams
  - **Face\_Mask** → a product type that contains all the products that are face masks
  - **Treatment** → a product type that contains all the products that are treatments
  - **Sun Protection** → a product type that contains all the products that are sun protections
- **ProductPurpose** → a class of product purposes
  - **Moisturizing** → A purpose for a product that can moisturize
  - **Soothing** → A purpose for a product that can soothe
  - **UV\_Protection** → A purpose for a product that has UV protection
  - **Purifying** → A purpose for a product that can purify
  - **Eye\_care** → A purpose for a product that can care for the eye

**Object Properties:**

- hasPurpose
- hasIngredient
- isOfBrand (potentially isBrandOf inverse)
- isBrandOf
- isIngredientOf
- hasComponent (imported from product ontology)

| Object type property | Relation   |
|----------------------|--|
| hasPurpose           | ProductType <i>hasPurpose</i> ProductPurpose<br>ProductName <i>hasPurpose</i> ProductPurpose |
| hasIngredient        | ProductName <i>hasIngredient</i> ProductIngredient   |
| isOfBrand            | ProductName <i>isOfBrand</i> ProductBrand  |
| isBrandOf            | ProductBrand <i>isBrandOf</i> ProductName  |
| isIngredientOf       | ProductIngredient <i>isIngredientOf</i> ProductName  |
| hasComponent         | ProductName <i>hasComponent</i> ProductIngredient  |

Table 1.

As can be seen in Table 1, the object properties connect the classes and the specific instances to each other.

**Data type properties:**

- hasPrice
- hasRating
- hasName
- isForCombination
- isForDry
- isForOily
- isForSensitive
- isForNormal

| Data type properties | Relation  |
|----------------------|---|
| hasPrice             | ProductName <i>hasPrice</i> “...”                                     |
| hasRating            | ProductName <i>hasRating</i> “...”                                    |
| hasName              | ProductName <i>hasName</i> “...”<br>ProductBrand <i>hasName</i> “...” |

|                  |   |
|------------------|---|
| isForCombination | ProductForCombination <i>isForCombination</i> “...” |
| isForDry         | ProductForDry <i>isForDry</i> “...”                 |
| isForOily        | ProductForOily <i>isForOily</i> “...”               |
| isForSensitive   | ProductForSensitive <i>isForSensitive</i> “...”     |
| isForNormal      | ProductForNormal <i>isForNormal</i> “...”           |

Table 2.

As can be seen in Table 2, the datatype properties connect the instances of classes to a particular literal, which could be a string or a number in our ontology. The datatype properties are explained below.

- **hasRating** data type property connects the Product to its rating, thus a positive integer. For instance, Crème de la Mer hasRating “8”.
- **hasName** connects the ProductBrand to its name. For instance, moisturizer hasName Crème de la Mer.
- **hasPrice** connects the ProductName to its price in USdollars. For instance, Crème de la Mer hasPrice \$175.
- **isForCombination** connects the class ProductForCombination to an integer either 0 or 1. This is because in our dataset a 1 means that the product is for combination skin and a 0 means that it does not belong to the class ProductForCombination.
- **isForDry** connects the class ProductForDry to an integer either 0 or 1. This is because in our dataset a 1 means that the product is for dry skin and a 0 means that it does not belong to the class ProductForDry.
- **isForOily** connects the class ProductForOily to an integer either 0 or 1. This is because in our dataset a 1 means that the product is for oily skin and a 0 means that it does not belong to the class ProductForOily.
- **isForSensitive** connects the class ProductForSensitive to an integer either 0 or 1. This is because in our dataset a 1 means that the product is for sensitive skin and a 0 means that it does not belong to the class ProductForSensitive.
- **isForNormal** connects the class ProductForNormal to an integer either 0 or 1. This is because in our dataset a 1 means that the product is for normal skin and a 0 means that it does not belong to the class ProductForNormal.

## **9. Ontology, class restrictions, and other existing ontologies**

For the ontology, we used the classes and properties we described in the conceptualization. We used these classes to map and link these classes and properties to others in external reusable ontologies. The two external ontologies we reused are the schema.org [1] ontology and the product vocabulary. The schema.org ontology contains relevant classes such as ProductName, Rating and ProductBrand. The class ProductBrand will be useful to connect the skincare products to their brand. To reuse only the classes we need, we copied the classes we need for our skincare ontology to a separate ontology. After doing this, we imported this piece of the schema.org ontology into our skincare ontology. The second vocabulary we reused is the Product [8] Vocabulary. We used the class Component from this vocabulary. The Component class is similar to ProductIngredient, which will be used to relate the ingredients to the ProductName.

The 17 class restrictions we made are:

### **1. ProductName hasComponent some ProductIngredient**

In the external dataset are the ingredients for each product, with this restriction we will relate the productName to its ingredient.

### **2. ProductName isOfBrand some ProductBrand**

In the external dataset are the brands for each productName, with this restriction we will relate the productName to its brand.

### **3. Component isIngredient some ProductName**

In the external dataset are ingredients listed for each product, with this restriction we will relate the component to their productName.

### **4. ProductBrand isBrandOf some ProductName**

In the external dataset are ProductNames for each brand, with this restriction we will relate the ProductBrand to each ProductName.

### **5. ProductIngredient isIngredient some ProductName**

In the external dataset are ingredients listed for each product, with this restriction we will relate the ingredient to their productName.

### **6. Cleanser hasPurpose some Purifying**

Cleanser has the purpose to purify, with this restriction we will relate the purpose purifying to each cleanser in the dataset.

### **7. Eye\_cream hasPurpose some Eye\_care**

Eye cream has the purpose eye care, with this restriction we will relate the purpose eye care to each eye cream in the dataset.

### **8. Face\_mask hasPurpose some Soothing**

Facemask has the purpose of soothing, with this restriction we will relate the purpose soothing to each face mask in the dataset.

### **9. Moisturizer hasPurpose some Moisturizing**

Moisturizer has the purpose to moisturize, with this restriction we will relate the purpose moisturizing to each moisturizer in the dataset.

### **10. Sun\_Protection hasPurpose some UV\_Protection**

Sun protection has the purpose to protect from UV, with this restriction we will relate the purpose UV\_protection to each Sun\_Protection in the dataset

### **11. isBrandOf inverse of isOfBrand**

If a product is of a brand, then that means that the brand is the brand of the product.

**12. hasComponent inverse of isIngredientOf**

If a product has a component, then that means that the component is the ingredient of the product.

**13. ProductForCombination isForCombination value 1**

If a product is for combination skin then that means that the data property isForCombination has value 1.

**14. ProductForDry isForDry value 1**

If a product is for dry skin then that means that the data property isForDry has value 1.

**15. ProductForOily isForOily value 1**

If a product is for oily skin then that means that the data property isForOily has value 1.

**16. ProductForNormal isForNormal value 1**

If a product is for normal skin then that means that the data property isForNormal has value 1.

**17. ProductForSensitive isForSensitive value 1**

If a product is for sensitive skin then that means that the data property isForSensitive has value 1.

**10. Integrated external datasets**

For the integration of external data, it is important to first note what form the data that is used is in. The application uses various datasets of different formats. The first dataset retrieved from Kaggle is a CSV file while the second dataset from DBPedia is in RDF-format [4].

The CSV file contains a large range of products together with their specific properties which are the type of product (label), the skin type it is suited for, and its brand, name, price, rating, and ingredients. Each row contains all the information about one product, where all of the previously mentioned properties are displayed in its own column.

As the CSV file is in tabular form, it has to be converted to TTL format before the useful information can be integrated in the application's ontology. In order to do so, GraphDB is used alongside its OntoRefine tool. The cosmetics dataset from Kaggle contains columns of which all are relevant to the application and so all columns are maintained. Every value is then added to the ontology as an instance of the corresponding class. Take as an example the class *ProductBrand* in the ontology which contains all instances from the table column *Brand*, so *Brand a ProductBrand*. In addition to classes, some properties (e.g. price and rating) will also be converted to literals to simplify the later process of finding matching instances or data. Furthermore, mappings will be created between the data in the table using the object properties defined in the ontology. An example of this would be *Name (a ProductName) isOfBrand Brand (a ProductBrand)*. As for the third potential chemical ingredients dataset [3], the only useful column will be the one containing the names of chemicals. These chemicals can be added as instances of the class *ProductIngredient* after which we can retrieve useful information about them from DBPedia that can then be presented to the user.

All DBPedia information will be acquired through multiple SPARQL queries using the DBPedia SPARQL endpoint in GraphDB. All necessary additional information about brands, individual products, types of products, or the ingredients used in products can be retrieved by creating SPARQL queries based on the instances that were added before. This additional information will be added to the ontology, in the form of triples, using an INSERT query.



## 11. Meaningful inferences in Protégé

We have provided the ontology with certain class restrictions, which resulted in the ontology having meaningful inferences. The following inferences were deducted:

- The first restriction results in having inferences in the individual of ProductBrand “Youth to the people”:

*Brand is BrandOf some ProductName*

This restriction states that a brand is the brand of some product. This is evidenced by two screenshots, where the first image shows the brand without the reasoner and the second image displays the brand with the ProductNames with the reasoner turned on. These screenshots can be found in the **Appendix B as figure 1.1 and figure 1.2**. As seen in *figure 1.2* with the reasoner turned on, it shows that the brand “Youth to the people” is a brand of a list of products.

- This second restriction results in having inferences in the class of ProductForOily:

*ProductForOily isForOily value 1*

This restriction states that every instance which is related to the integer value 1 using the data property *isForOily* will be regarded as a product for oily skin. This is evidenced by two screenshots, where the first image shows the ProductForOily with the reasoner turned off and the bottom image displays the brand with the ProductNames with the reasoner turned on. These screenshots can be found in the **Appendix B as figure 2.1 and figure 2.2**. As seen in *figure 2.2* with the reasoner turned on, it shows that the class ProductForOily has a list of products that is for oily skin, this is because these products had the value 1 for the data property *isForOily*.

- The following restriction results in having inferences in the individual of ProductName “City\_Sunscreen\_Serum\_SPF\_30”:

*Sun\_Protection hasPurpose some UV\_Protection*

This restriction states that an instance that has purpose UV\_Protection is a Sun\_Protection. This is evidenced by two screenshots, where the first image shows the product without the reasoner and the second image displays the product with the reasoner turned on. These screenshots can be found in the **Appendix B as figure 3.1 and figure 3.2**. As seen in *figure 3.2* with the reasoner turned on, it shows that City\_Sunscreen\_Serum\_SPF\_30 is of type Sun\_Protection.

## 12. Complex SPARQL queries

For our skincare application, we designed multiple SPARQL queries that retrieve results from the ontology and integrated data. The query that is used depends on what function of the application is being executed by the user. These queries look as follows;

### SPARQL query 1 - Recommendation engine:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX skincare: <http://www.semanticweb.org/skincare/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX schema: <https://schema.org/>
```

```
SELECT ?name ?brand ?ingredients ?priceInUSD ?rating WHERE{
    ?product a schema:ProductName;
    a skincare:Treatment;
    skincare:hasName ?name;
    skincare:isOfBrand ?skincareBrand;
    skincare:hasIngredients ?ingredients;
    skincare:hasPrice ?priceInUSD;
    skincare:hasRating ?rating;
    skincare:isForCombination 1.
    FILTER(?priceInUSD < 50 && ?rating > 4)

    ?skincareBrand skincare:hasName ?brand .
}
```

Query 1 retrieves information from DBpedia about different products that were added as instances from the Kaggle CSV file. This is the main function of the skincare application. The user gives their input on the small questionnaire at the start which will contain the values that are used for **the product type, skin type, price**, and so on. Depending on their input, the SPARQL query gives them a table that contains the results for the query. For query one the example input was chosen as follows:

- The type of product the user is looking for is a treatment (skincare:Treatment)
- The user wants a product suitable for combination skin (skincare:isForCombination “1”)
- The user wants to pay a price of up to 80 USD and they want the product to have a rating higher than 4.5

The output that the user gets from the application shows the name of the selected products, alongside their brands, ingredients, prices in US dollars, and ratings.

**SPARQL query 2 - Get additional information about ingredients:**

**PREFIX rdf:** <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

**PREFIX skincare:** <http://www.semanticweb.org/skincare/>

**PREFIX rdfs:** <http://www.w3.org/2000/01/rdf-schema#>

**PREFIX owl:** <http://www.w3.org/2002/07/owl#>

**PREFIX dbo:** <http://dbpedia.org/ontology/>

**SELECT DISTINCT ?label ?comment ?alternativeNames ?link WHERE{  
    ?ingredient a skincare:ProductIngredient;  
    skincare:hasName ?label.**

**OPTIONAL {SERVICE <https://dbpedia.org/sparql> {  
    ?dbpingredient rdfs:label ?label.  
    OPTIONAL{?dbpingredient rdfs:comment ?comment.  
        FILTER(lang(?comment) = 'en')}  
    OPTIONAL{?dbpingredient dbo:alternativeName ?alternativeNames.}  
    OPTIONAL{?dbpingredient dbo:wikiPageExternalLink ?link}  
    }  
}**

This QUERY obtains more information about the ingredients that can be found in products. When a user gets products recommended, and even when they already own some products, they might wonder what the ingredients used in these products are. To many customers, knowing what is in the products they buy is something of great importance. Take for instance vegans, or people who are highly sensitive to all sorts of chemicals, for these people this function will be a huge help in selecting the right skincare items as it gives them more insight in how the ingredient is obtained, et cetera.

For query 2 the example input was chosen as follows:

- The user is looking for the ingredients (skincare:ProductIngredient)
- The user wants the ingredients for a certain product (skincare:hasName)

The output the user gets from the application shows the name of the ingredient, a description of the ingredient, an alternative name, and a link to the ingredient.

### **SPARQL query 3 - Get additional information about skincare brands:**

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX skincare: <http://www.semanticweb.org/skincare/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX schema: <https://schema.org/>
```

```
SELECT DISTINCT ?label ?comment ?founder ?foundingYear ?products ?link WHERE{
    ?brand a schema:ProductBrand;
    skincare:hasName ?label.
```

```
    OPTIONAL {SERVICE <https://dbpedia.org/sparql> {
        ?dbpbrand rdfs:label ?label.
        OPTIONAL{?dbpbrand rdfs:comment ?comment.
            FILTER(lang(?comment) = 'en')}}
        OPTIONAL{?dbpbrand dbo:foundedBy ?name.
            ?name rdfs:label ?founder.
            FILTER(lang(?founder) = 'en')}}
        OPTIONAL{?dbpbrand dbo:foundingYear ?foundingYear.}
        OPTIONAL{?dbpbrand dbp:products ?products.}
        OPTIONAL{?dbpbrand dbp:homepage ?link.}
    }}
}
```

Query 3 shows how additional information about brands will be retrieved using the ontology and external data from DBPedia. The properties on which it searched would be based on the user's input about which brand they want to know more about. These options are limited to the brands that our application uses and which also have their own DBPedia page as this is needed for the extra information. This function is considered useful as people often want to know whether a brand is trustworthy, in line with its principles, and of good quality. The application will only provide descriptions and some general information which should help the user when searching for more information by themselves.

For query 3 the example input was chosen as follows:

- The user wants to know about a brand (skincare:ProductBrand)

The output that the user gets from the application shows the name of the brand, a description of the brand, the founder of the brand, the founding year of the brand, and a link to the brand website and their products.

### 13. Inferences of the SPARQL queries

To show that the ontology indeed produces inferences, the following rules from section 9 (Ontology, class restrictions, and other existing ontologies) were used for this:

- Rule 1: *ProductName* hasComponent some *ProductIngredient*
- Rule 15: *ProductForOily* isForOily value 1

For rule 1, the reasoner has been turned off in GraphDB's SPARQL function. It can be seen that no data is the result of the query, see **Appendix C, figure 1.1**. This is caused by rule 1 as it states that *ProductName* hasComponent some *ProductIngredient*. The reasoner infers that *Component EquivalentTo ProductIngredient* as both contain the same necessary and sufficient restriction *isIngredientOf some ProductName*. As a result, when it is known that a *ProductName* has a component which is an instance, it infers that this instance is of the class *ProductIngredient*, and as this class is equivalent to the *Component* class, it is inferred that the instance is a *Component*. For the full screenshot of this inference, see **Appendix C, figure 1.2**.

Considering rule 15, *ProductForOily* isForOily value 1, it means that every instance which is related to the integer value 1 using the data property *isForOily* will be regarded as a product for oily skin. As this is a conclusion based on inferencing, the SPARQL query with the reasoner off returns no data, see **Appendix C, figure 2.1**. If the reasoner is turned on, a list will appear of all products suitable for oily skin as they have the value 1 for the data property *isForOily*, see **Appendix C, figure 2.2**.

14. See the ipynb files present in the submitted zip for a representation of the application and visualizations we use.

### References:

1. <https://schema.org/Product>
2. <http://ns.inria.fr/provoc>
3. <https://www.kaggle.com/kingabzpro/cosmetics-datasets>
4. <https://www.dbpedia.org/>
5. <https://dbpedia.org/page/Clinique>
6. [https://dbpedia.org/page/Category:Skin\\_care](https://dbpedia.org/page/Category:Skin_care)
7. <https://dbpedia.org/sparql>
8. [http://ns.inria.fr/provoc/v1/provoc\\_v1.html](http://ns.inria.fr/provoc/v1/provoc_v1.html)

## Appendix A

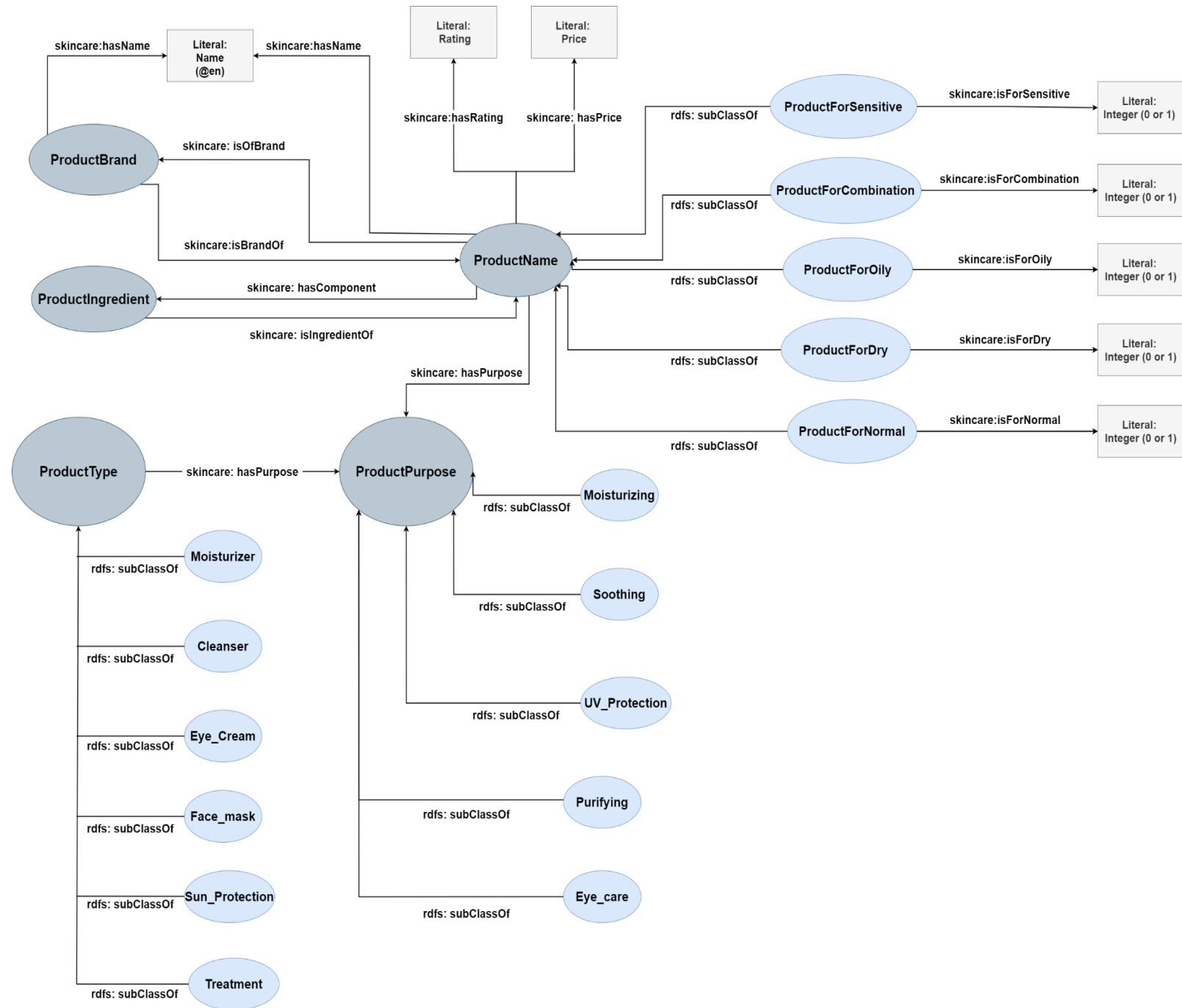


Figure 1. The graph of the created skincare ontology

## Appendix B

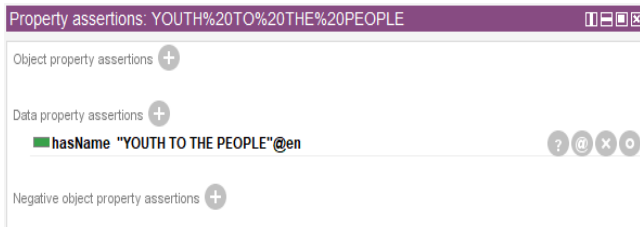


Figure 1.1: ProductBrand instance equivalence with reasoning turned off

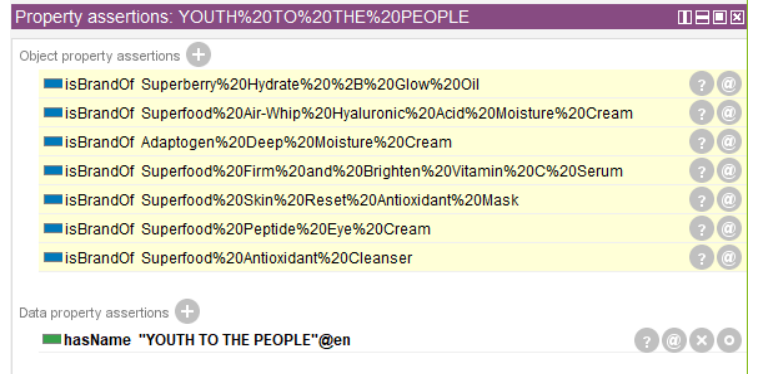


Figure 1.2: ProductBrand instance equivalence with reasoning turned on

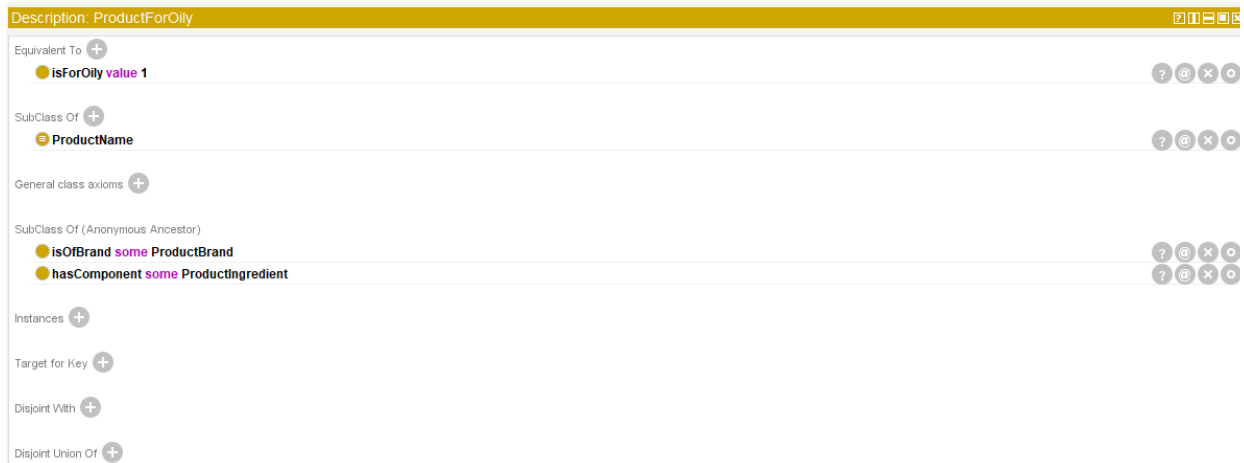
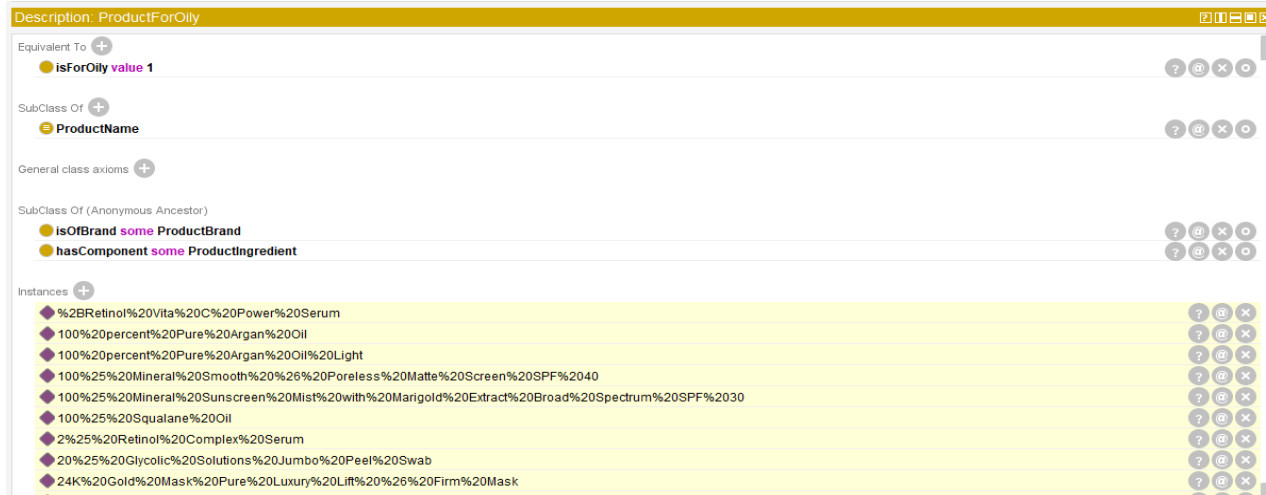


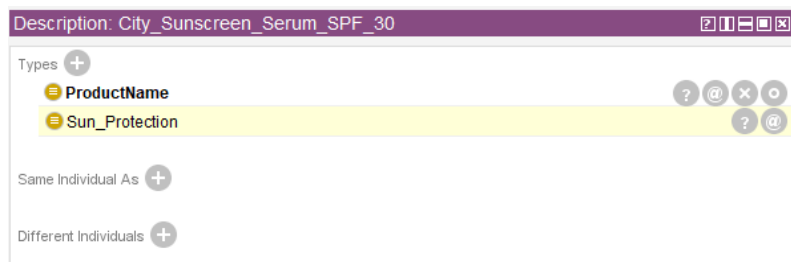
Figure 2.1: ProductForOily class equivalence with reasoning turned off



**Figure 2.2: ProductForOily class equivalence with reasoning turned on**



**Figure 3.1: Instance of ProductName equivalence with the reasoner turned off**



**Figure 3.2: Instance of ProductName equivalence with the reasoner turned on**



## Appendix C

The screenshot shows a query editor with the following query:

```
1 PREFIX rdt: <http://www.w3.org/1999/02/22-rdt-syntax-ns#>
2 PREFIX skincare: <http://www.semanticweb.org/skincare/>
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4 PREFIX owl: <http://www.w3.org/2002/07/owl#>
5 PREFIX dbo: <http://dbpedia.org/ontology/>
6 PREFIX dbp: <http://dbpedia.org/property/>
7 PREFIX schema: <https://schema.org/>
8 PREFIX pv: <http://ns.inria.fr/provoc#>
9
10 SELECT DISTINCT * WHERE{
11   ?ingredient a pv:Component
12 }
```

The interface includes a 'Run' button and a 'Download as' button. Below the query, there are tabs for 'Table', 'Raw Response', 'Pivot Table', and 'Google Chart'. A message states: 'No results. Query took 0.1s, moments ago.' Below this, a table header 'ingredient' is shown, followed by the message 'No data available in table'.

**Figure 1.1 Inferencing, rule 1: ProductName hasComponent some Component (reasoning off)**

The screenshot shows the same query as Figure 1.1, but with the following query:

```
4 PREFIX owl: <http://www.w3.org/2002/07/owl#>
5 PREFIX dbo: <http://dbpedia.org/ontology/>
6 PREFIX dbp: <http://dbpedia.org/property/>
7 PREFIX schema: <https://schema.org/>
8 PREFIX pv: <http://ns.inria.fr/provoc#>
9
10 SELECT DISTINCT * WHERE{
11   ?ingredient a pv:Component.
12 }
```

The interface includes a 'Run' button and a 'Download as' button. Below the query, there are tabs for 'Table', 'Raw Response', 'Pivot Table', and 'Google Chart'. A message states: 'Showing results from 1 to 1,000 of 7,168. Query took 0.2s, moments ago.' Below this, a table header 'ingredient' is shown, followed by the following results:

|   | ingredient                        |
|---|-----------------------------------|
| 1 | skincare:Charcoal                 |
| 2 | skincare:Glycerin                 |
| 3 | skincare:Sodium_lauroyl_glutamate |
| 4 | skincare:Stearic_Acid             |
| 5 | skincare:Water                    |

**Figure 1.1 Inferencing, rule 1: ProductName hasComponent some Component (reasoning on)**

Unamed x Unamed x Unamed x Unamed x Unamed x + KD-Project v

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX skincare: <http://www.semanticweb.org/skincare/>
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4 PREFIX owl: <http://www.w3.org/2002/07/owl#>
5 PREFIX dbo: <http://dbpedia.org/ontology/>
6 PREFIX dbp: <http://dbpedia.org/property/>
7 PREFIX schema: <https://schema.org/>
8 PREFIX pv: <http://ns.inria.fr/provoc#>
9
10 SELECT DISTINCT * WHERE{
11   ?product a skincare:ProductForOily.
12 }
```

Run

Press Alt+Enter to autocomplete

Table Raw Response Pivot Table Google Chart Download as v

Filter query results No results. Query took 0.1s, moments ago.

|                            | product |
|----------------------------|---------|
| No data available in table |         |

**Figure 2.1 Inferencing, rule 15: ProductForOily isForOily value 1 (reasoning off)**

Unamed x Unamed x Unamed x Unamed x Unamed x + KD-Project v

```
5 PREFIX dbo: <http://dbpedia.org/ontology/>
6 PREFIX dbp: <http://dbpedia.org/property/>
7 PREFIX schema: <https://schema.org/>
8 PREFIX pv: <http://ns.inria.fr/provoc#>
9
10 SELECT DISTINCT * WHERE{
11   ?product a skincare:ProductForOily.
12 }
```

Run

Press Alt+Enter to autocomplete

Table Raw Response Pivot Table Google Chart Download as v

Filter query results Showing results from 1 to 894 of 894. Query took 0.2s, moments ago.

|   | product  |
|---|--|
| 1 | skincare:Cleanser  |
| 2 | skincare:Crème%20de%20la%20Mer   |
| 3 | skincare:Facial%20Treatment%20Essence                                      |
| 4 | skincare:Protini™%20Polypeptide%20Cream                                    |
| 5 | skincare:The%20Moisturizing%20Soft%20Cream                                 |
| 6 | skincare:Your%20Skin%20But%20Better™%20CC%2B™%20Cream%20with%20SPF%2050%2B |

**Figure 2.2 Inferencing, rule 15: ProductForOily isForOily value 1 (reasoning off)**