The plant, I want

Ruud van Bakel, Janus Jansen, Lieke Venneker

Group 06

Abstract. A lot of people want plants in their direct environment, be it in their house for decoration, or in the office to clean the air a bit. When deciding to get a plant, it might be very difficult to pick the perfect plant, fitting the requirements. Every plant has different needs, looks and special features. The application “The plant, I want” can help with choosing the ideal plant, by listing different possible qualities and needs of plants. The user of the application can then choose what their wants and needs are, and “The plant, I want” generates a list with suitable plants, with additional information about them. The system works based on an ontology and uses an external sparql endpoint. “The plant, I want” is an excellent consultant when looking for a plant.

1 Introduction

To enrich a space, a great option is to decorate it with plants. Plants are often very durable, and fit in almost every environment. Plants can enrich a room, filter the air and add aesthetic quality. The problem is, there are a lot of plants to choose from. Some require a lot of care, with regular watering and a specific temperature. Other plants barely need any attention, and are thus great for a very different needs.

To make the choice of plants a lot easier, the application “The plant, I want” was created. The application revolves around plants, more specifically plants that can be bought by Dutch consumers for in their house, office or outside area. The application has a range of different kind of plants for the consumer to choose from, for example a plant that needs little to no attention and can be placed in any kind of environment and needs no specific humidity. “The plant, I want” will help the users find the perfect plant for them, by giving options to select which will lead to the display of different plants that fit the requirements of the user of the application. This way, everyone can use the application: from experts in gardening to a someone who wants to decorate their living room.

Knowledge and Data 2018
Final Assignment

2 External datasets

As external datasets for the application, two sources were used. The first one is DBpedia. DBpedia was chosen because it has a lot of information, and has a really nice endpoint, which was used. DBpedia also has a lot of plants and information about them, which makes it ideal to use for the application.

Secondly, a csv-file "San Francisco Plant Finder Data"[1] was used. This file was transformed into an rdf-file using OpenRefine. An rdf-file of the San Francisco Plant Finder Data was also available, but when opened in Protege, there were no classes and no entities, only annotation properties were visible. Because the offered rdf-file was not useful for the application but the csv-file did have interesting information, the decision to transform an csv-file was made. With the transformation, a lot of the information of the csv-file was used, and unnecessary information for this project was left out. The usage of this file led to a lot of information, multiple classes, several properties and a lot of entities, it really enriched the application. All the classes, entities and properties got the prefix pl, so it could be added to the basic ontology without any problems.

3 Vocabularies and ontologies

The basis ontology uses pl as prefix, which is derived from plant, and refers to http://semanticweb.org/ruud/BetterPlants/. On top of this vocabulary, five already existing and established ones were used. The first one being dbo, which refers to http://dbpedia.org/ontology/. This covers dbo:family, dbo:thumbnail, dbo:abstract and dbo:Plant. The second used vocabulary has the prefix dbp; this comes from http://dbpedia.org/property/. This is where the dbp:binomial comes from. The third used vocabulary comes from http://xmlns.com/foaf/0.1/, with the prefix foaf. Foaf stands for 'friend of a friend', and is often used to describe relations between persons and/or objects. In the application, foaf is used as foaf:isPrimaryTopicOf. This is used to get a wikipedia link form plants found on dbpedia. The fourth used vocabulary comes from http://umbel.org/umbel/rc/ , and has the prefix umbel-rc. It is used for the class umbel-rc:Plant. The fifth and final external vocabulary has the prefix yago, which comes from http://dbpedia.org/class/yago/ . It is used as yago:Plant100017222 , which is a class for plants.

[1] https://catalog.data.gov/dataset/sf-plant-finder-database

4 Building the ontology

As base of the ontology the class plant was chosen, since the ontology revolves around choosing plants to buy. Because the application will help with the purchase of plants, it contains a lot of information about different plants. For the user of “The plant, I want” it would be useful to know how the plant needs to be cared for, which include the light requirement (low, medium, high) and the water requirement (low, medium, high). Because the plants are going to be in or around the house or office of the person searching for a suitable plant, users might also find it interesting to know the leaf color of the plant. For extra information the latin and dutch name were added to the ontology, which means that the plant will be found when either of those names are used.

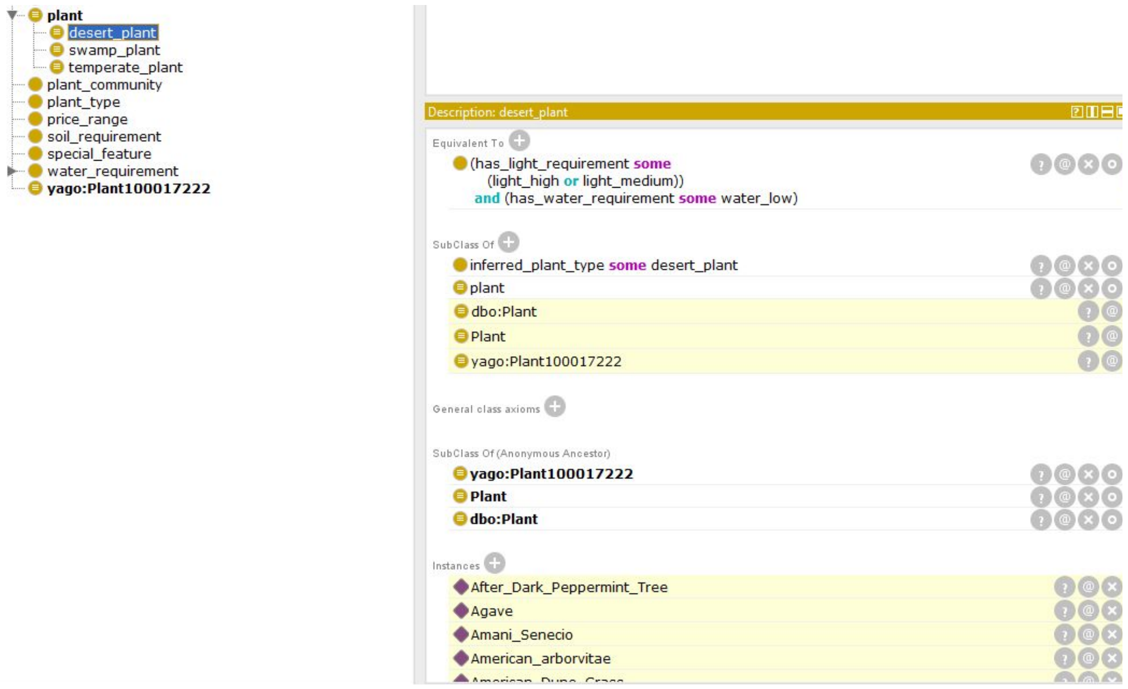
From the file “San Francisco Plant Finder Data”, which was transformed into rdf, the classes pl:appropriate_location, pl:associated_widlife, pl:bloom_time, pl:family_name, pl:flower_colour, pl:light_requirement, pl:plant, pl:plant_community, pl:plant_type, pl:soil_requirement, pl:special_feature and pl:water_requirement were added, or the classes that already existed with the same information were complemented. The addition of this rdf-file also produced a lot of instances, namely plants. Besides the classes and instances, the data-properties pl:additional_information, pl:has_common_name, pl:has_latin_name and pl:has_size_at_maturity came into existence, just like the object-properties pl:attracts_wildlife, pl:has_appropriate_location, pl:has_bloom_time, pl:has_flower_color, pl:has_leaf_color, pl:has_light_requirement, pl:has_plant_community, pl:has_soil_requirement, pl:has_water_requirement, pl:is_of_family and pl:is_plant_type. This enriched the ontology, and made it a lot more useful. Because of the file “San Francisco Plant Finder Data”, there are a lot of individual plants in the ontology, and a lot of features of the plants are present. This is very useful when picking a suitable plant, because now the needs and qualities of the plants are available in one place. The ontology also makes use of dbo:Plant, umbel-rc:Plant and yago:Plant100017222. These three are classes that we use in the querying of the dbpedia data. Every plant we get from dbpedia has to be a dbo:Plant, umbel-rc:Plant and yago:Plant100017222, to make sure (almost) all results are plants. In the ontology, these classes all have an equivalence relation te each other. These classes are also in an equivalence relation with the class pl:plant. With this, the plants from the transformed csv-file and the plant found at dbpedia are both considered to be of the same type. An important property, that was also derived from dbpedia, was dbp:binomial.

The ontology was supposed to map the plants from both datasets using an owl:InverseFunctionalProperty relation. This relation would be applied to pl:has_latin_name and dbp:binomial. pl:latin_name and dbp:binomial would also said to be equivalent to each other. With this, the ontology should combine all plants from both datasets with the same latin name. Unfortunately, protege didn't allow for a owl:dataTypeProperty to have an owl:InverseFunctionalProperty relation, so this mapping could not be done.

Furthermore, the ontology also specifies other properties, used by dbpedia. These properties are dbo:abstract, dbo:thumbnail, dbo:family and foaf:isPrimaryTopicOf. These properties were used for getting relevant data about the plants queried from dbpedia, to show in the application to the user.

5 Inferences

Although the ontology doesn't infere between the different datasets, some other simple inferencing is done. The ontology does infer plants to be desert plants, swamp plants or temperate plants, based on their light requirement and water requirement, see figures 1, 2 and 3. It is assumed that if a plant has a light requirement belonging to either the class light_high or light_medium and it has a water requirement belonging to the class water_low, the plant is a desert plant. This is assumed, because desert plants typically get a lot of light and don't require much water. On the other had, if a plant a light requirement belonging to either the class light_low or light_medium and has a water requirement belonging to the class water_high, it is assumed that is it a swamp plant. This is assumed, because swamp plants typically live in watery environments and often don't get much light, because of other plants and trees blocking out the sun. Finally, plants with a light requirement belonging to either the class light_low or light_high and they have a water requirement belonging to the class water_medium, it is assumed that they are temperate plants. This is assumed, because plants in a temperate environment often don't get particularly much or little water and they often don't get particularly much sun. Temperate plants could however get little light when i a forrest for example, so light_low is also included.



plant
- desert_plant
- swamp_plant
- temperate_plant

plant_community
plant_type
price_range
soil_requirement
special_feature
water_requirement
yago:Plant100017222

Description: desert_plant

Equivalent To

(has_light_requirement some
  (light_high or light_medium))
 and (has_water_requirement some water_low)

SubClass Of

inferred_plant_type some desert_plant
plant
dbo:Plant
Plant
yago:Plant100017222

General class axioms

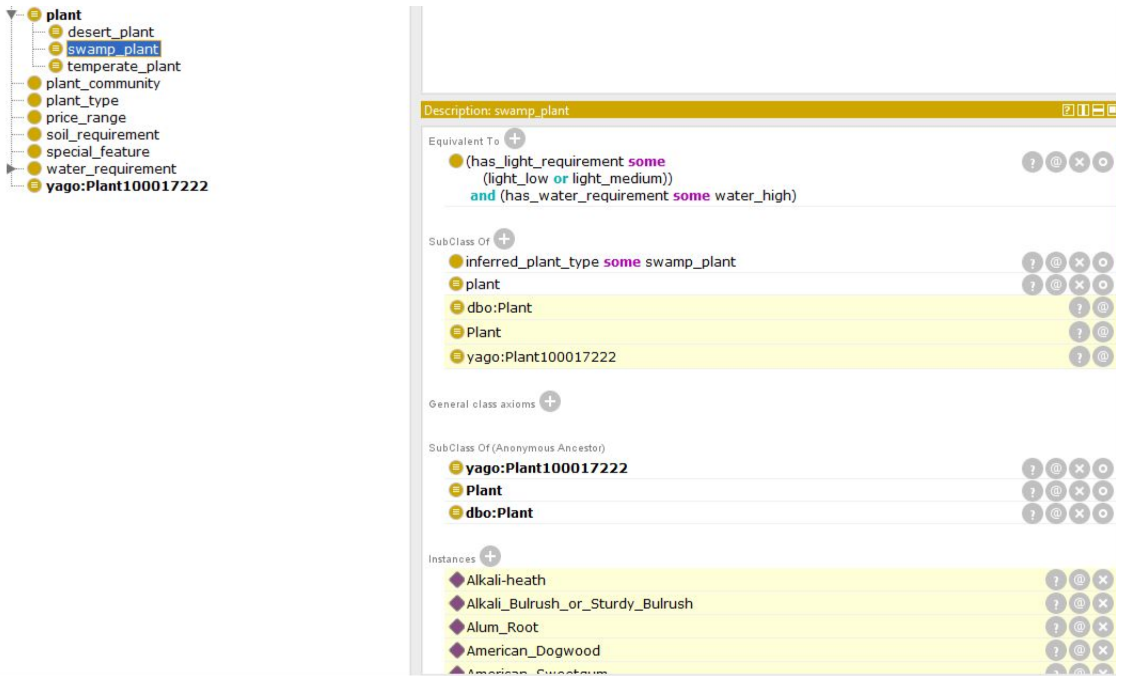
SubClass Of (Anonymous Ancestor)

yago:Plant100017222
Plant
dbo:Plant

Instances

After_Dark_Peppermint_Tree
Agave
Amani_Senecio
American_arborvitae

figure 1. Infered desert plants



plant
- desert_plant
- swamp_plant
- temperate_plant

plant_community
plant_type
price_range
soil_requirement
special_feature
water_requirement
yago:Plant100017222

Description: swamp_plant

Equivalent To

(has_light_requirement some
  (light_low or light_medium))
 and (has_water_requirement some water_high)

SubClass Of

inferred_plant_type some swamp_plant
plant
dbo:Plant
Plant
yago:Plant100017222

General class axioms

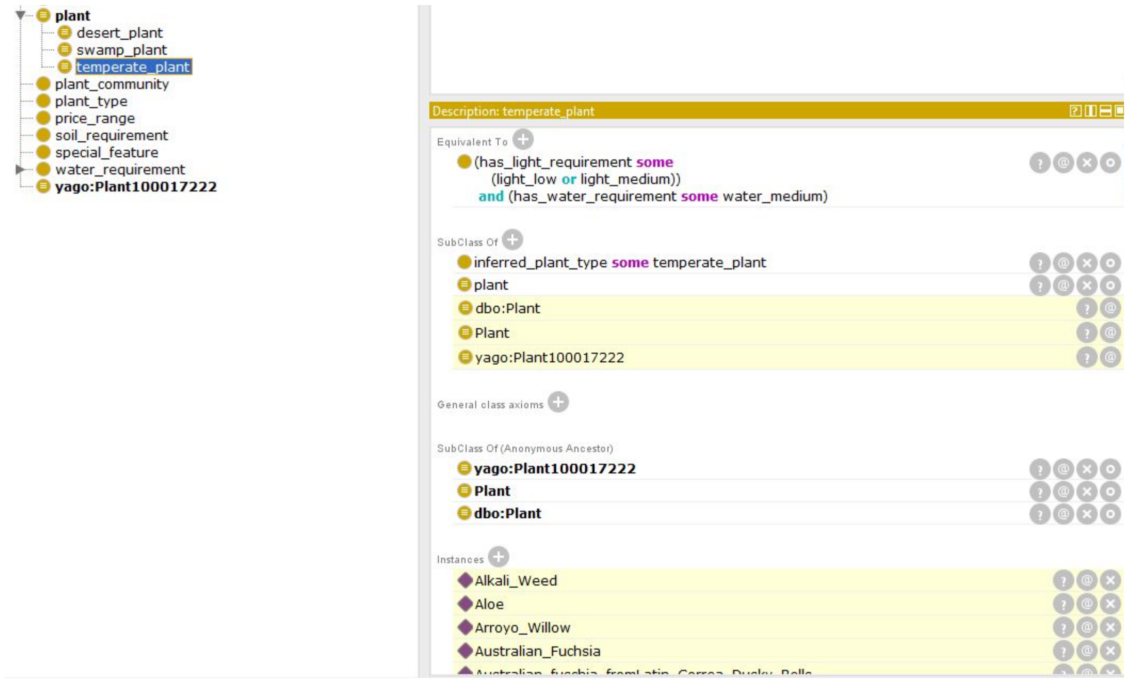
SubClass Of (Anonymous Ancestor)

yago:Plant100017222
Plant
dbo:Plant

Instances

Alkali-heath
Alkali_Bulrush_or_Sturdy_Bulrush
Alum_Root
American_Dogwood

figure 2. Infered swamp plants

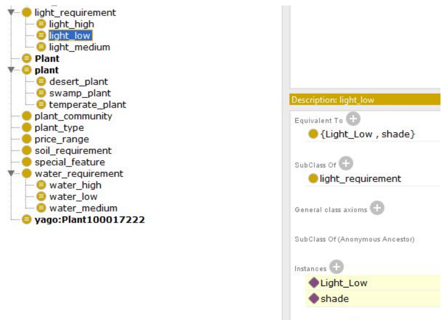


```
▼ plant
    desert_plant
    swamp_plant
    temperate_plant
  plant_community
  plant_type
  price_range
  soil_requirement
  special_feature
▶ water_requirement
  yago:Plant100017222
```

```
Description: temperate_plant

Equivalent To
  (has_light_requirement some
      (light_low or light_medium))
   and (has_water_requirement some water_medium)

SubClass Of
  inferred_plant_type some temperate_plant
  plant
  dbo:Plant
  Plant
  yago:Plant100017222

General class axioms

SubClass Of (Anonymous Ancestor)
  yago:Plant100017222
  Plant
  dbo:Plant

Instances
  Alkali_Weed
  Aloe
  Arroyo_Willow
  Australian_Fuchsia
```

figure 3. Infered swamp plants

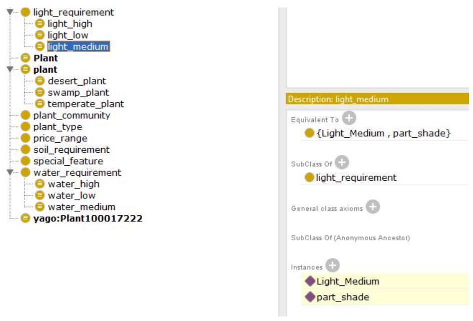
In order to use the data from the transformed csv-file to infer whether the plants for this dataset were deserts, swamp or temperate plants some other inferencing had to be done. The instances light_low, light_medium, light_high, water_low, water_medium and water_high were mapped to shade, part_shade, sun, none, low and moderate respectively, see figures 4, 5, 6, 7, 8 and 9.



```
▼ light_requirement
    light_high
    light_low
    light_medium
  Plant
▼ plant
    desert_plant
    swamp_plant
    temperate_plant
  plant_community
  plant_type
  price_range
  soil_requirement
  special_feature
▼ water_requirement
    water_high
    water_low
    water_medium
  yago:Plant100017222
```

```
Description: light_low

Equivalent To
  {Light_Low , shade}

SubClass Of
  light_requirement

General class axioms

SubClass Of (Anonymous Ancestor)

Instances
  Light_Low
  shade
```

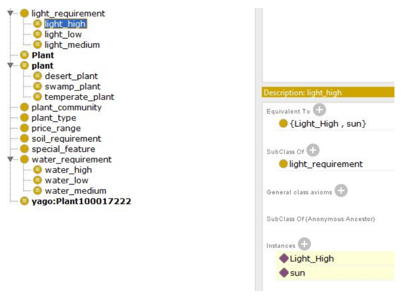
figure 4. light_low



```
▼ light_requirement
    light_high
    light_low
    light_medium
  Plant
▼ plant
    desert_plant
    swamp_plant
    temperate_plant
  plant_community
  plant_type
  price_range
  soil_requirement
  special_feature
▼ water_requirement
    water_high
    water_low
    water_medium
  yago:Plant100017222
```

```
Description: light_medium

Equivalent To
  {Light_Medium , part_shade}

SubClass Of
  light_requirement

General class axioms

SubClass Of (Anonymous Ancestor)

Instances
  Light_Medium
  part_shade
```

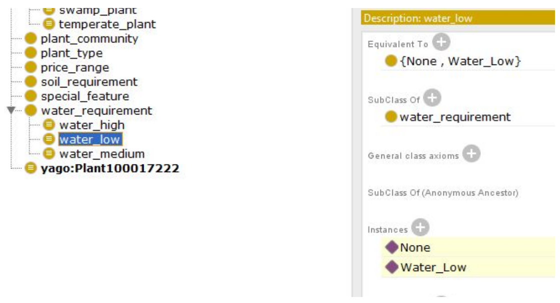
figure 5. light_medium



```
▼ light_requirement
    light_high
    light_low
    light_medium
  Plant
▼ plant
    desert_plant
    swamp_plant
    temperate_plant
  plant_community
  plant_type
  price_range
  soil_requirement
  special_feature
▼ water_requirement
    water_high
    water_low
    water_medium
  yago:Plant100017222
```

```
Description: light_high
Equivalent To
  {Light_High , sun}
SubClass Of
  light_requirement
General class axioms
SubClass Of (Anonymous Ancestor)
Instances
  Light_High
  sun
```

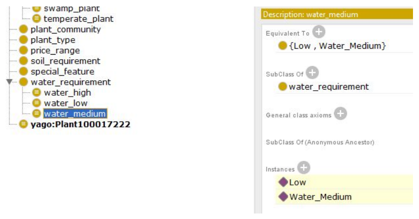
figure 6. light_high



```
    swamp_plant
    temperate_plant
  plant_community
  plant_type
  price_range
  soil_requirement
  special_feature
▼ water_requirement
    water_high
    water_low
    water_medium
  yago:Plant100017222
```

```
Description: water_low
Equivalent To
  {None , Water_Low}
SubClass Of
  water_requirement
General class axioms
SubClass Of (Anonymous Ancestor)
Instances
  None
  Water_Low
```

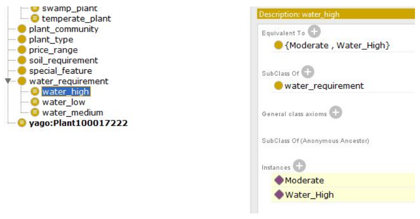
figure 7. water_low



```
    swamp_plant
    temperate_plant
  plant_community
  plant_type
  price_range
  soil_requirement
  special_feature
▼ water_requirement
    water_high
    water_low
    water_medium
  yago:Plant100017222
```

```
Description: water_medium
Equivalent To
  {Low , Water_Medium}
SubClass Of
  water_requirement
General class axioms
SubClass Of (Anonymous Ancestor)
Instances
  Low
  Water_Medium
```

figure 8. water_medium



```
    swamp_plant
    temperate_plant
  plant_community
  plant_type
  price_range
  soil_requirement
  special_feature
▼ water_requirement
    water_high
    water_low
    water_medium
  yago:Plant100017222
```

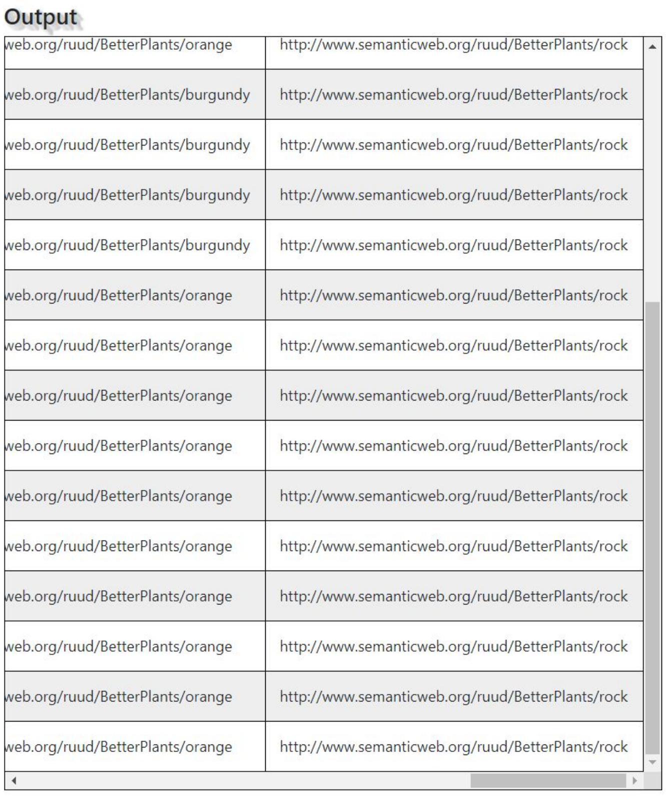
```
Description: water_high
Equivalent To
  {Moderate , Water_High}
SubClass Of
  water_requirement
General class axioms
SubClass Of (Anonymous Ancestor)
Instances
  Moderate
  Water_High
```

figure 9. water_high

6 Building the application

The first ideas that came up for the application were in hindsight way too big to complete in the timespan that was given. For example an idea that came up was an website with multiple areas that the user could scroll through with, for example, a desert where cacti were located and an swamp where other kinds of plants would live. The user could then click on these plants to get more information about them and the user would be able to compare them to other plants. When the first prototypes of the application were made it was immediately clear that this could not be done.

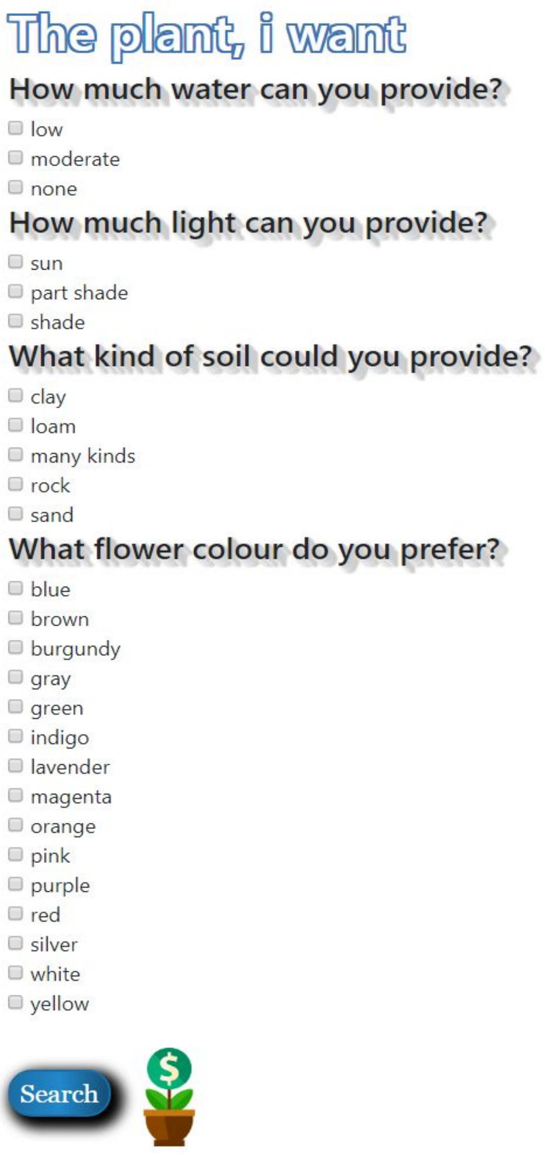
After realising this, the plan became to make the application working first, and visible appealing later. First the query's were made to know what kind of data was available and how this could be used. There is one query to get data from dbpedia and one query to get data from stardog, where the self-made ontology is hosted. Then the checkboxes were made so the user could select and filter the results from the query's. The input from the checkbox is put into an empty list, later in the code this list is iterated and put into an string with the appropriate surrounding text to be able to change the query. This happens when the search bar on the bottom of the page is clicked, then when the json data is returned this is being iterated and put into a table, which can be seen in figure 10. This table can become quite large if a big number of checkboxes are selected, this is why the table is fixed in size and scrollable both horizontal and vertical.



Output

web.org/ruud/BetterPlants/orange	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/burgundy	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/burgundy	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/burgundy	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/burgundy	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/orange	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/orange	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/orange	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/orange	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/orange	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/orange	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/orange	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/orange	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/orange	http://www.semanticweb.org/ruud/BetterPlants/rock
web.org/ruud/BetterPlants/orange	http://www.semanticweb.org/ruud/BetterPlants/rock

figure 10. Table with the output from a query



The plant, i want

How much water can you provide?

- ☐ low
- ☐ moderate
- ☐ none

How much light can you provide?

- ☐ sun
- ☐ part shade
- ☐ shade

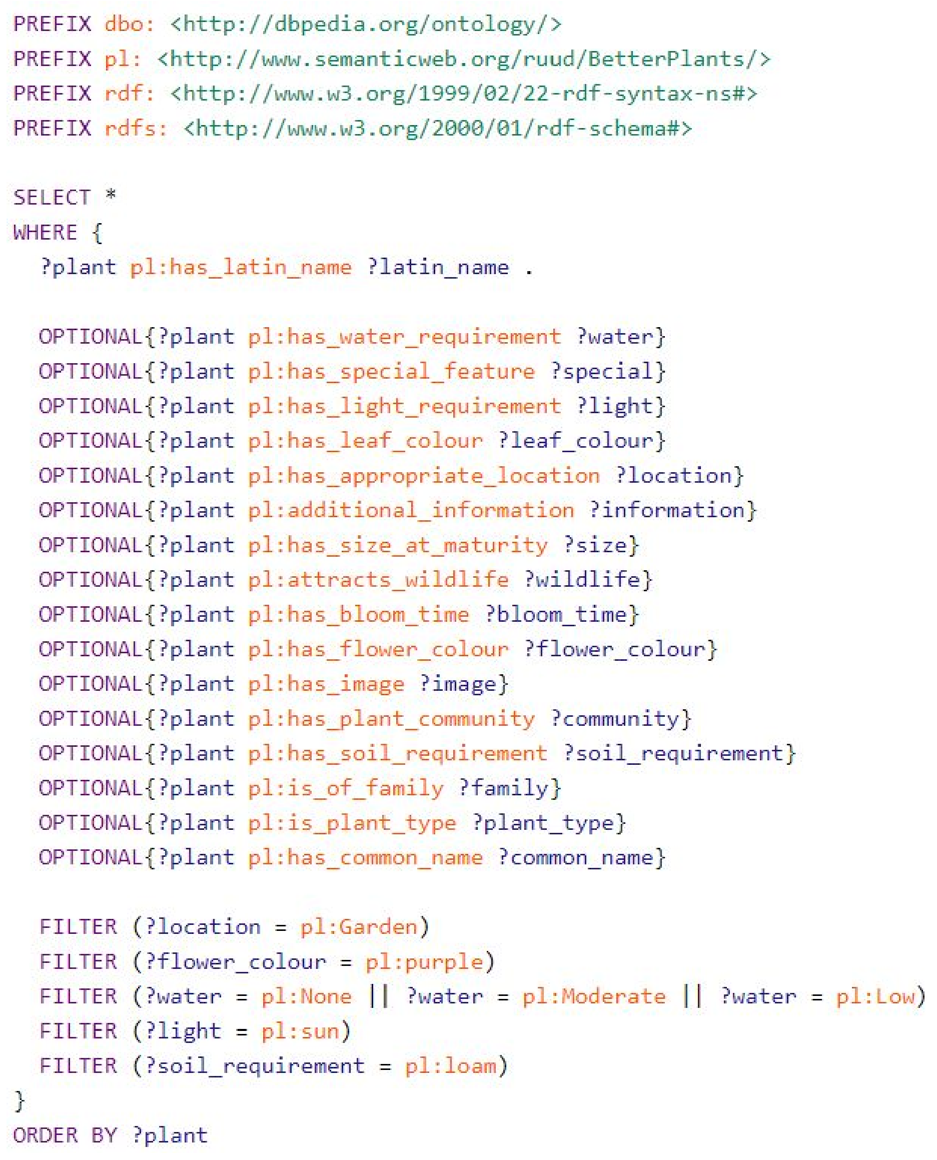
What kind of soil could you provide?

- ☐ clay
- ☐ loam
- ☐ many kinds
- ☐ rock
- ☐ sand

What flower colour do you prefer?

- ☐ blue
- ☐ brown
- ☐ burgundy
- ☐ gray
- ☐ green
- ☐ indigo
- ☐ lavender
- ☐ magenta
- ☐ orange
- ☐ pink
- ☐ purple
- ☐ red
- ☐ silver
- ☐ white
- ☐ yellow

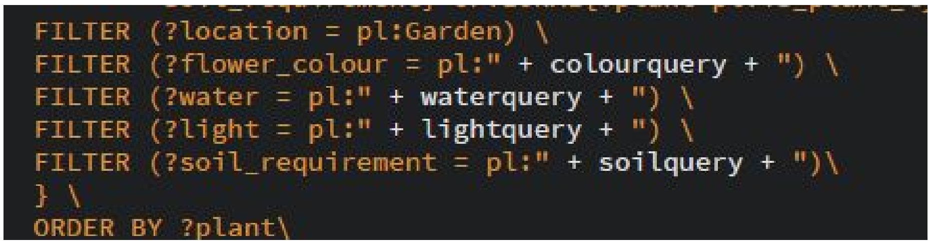
Figure 11. The options the user is greeted with



```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX pl: <http://www.semanticweb.org/ruud/BetterPlants/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT *
WHERE {
  ?plant pl:has_latin_name ?latin_name .

  OPTIONAL{?plant pl:has_water_requirement ?water}
  OPTIONAL{?plant pl:has_special_feature ?special}
  OPTIONAL{?plant pl:has_light_requirement ?light}
  OPTIONAL{?plant pl:has_leaf_colour ?leaf_colour}
  OPTIONAL{?plant pl:has_appropriate_location ?location}
  OPTIONAL{?plant pl:additional_information ?information}
  OPTIONAL{?plant pl:has_size_at_maturity ?size}
  OPTIONAL{?plant pl:attracts_wildlife ?wildlife}
  OPTIONAL{?plant pl:has_bloom_time ?bloom_time}
  OPTIONAL{?plant pl:has_flower_colour ?flower_colour}
  OPTIONAL{?plant pl:has_image ?image}
  OPTIONAL{?plant pl:has_plant_community ?community}
  OPTIONAL{?plant pl:has_soil_requirement ?soil_requirement}
  OPTIONAL{?plant pl:is_of_family ?family}
  OPTIONAL{?plant pl:is_plant_type ?plant_type}
  OPTIONAL{?plant pl:has_common_name ?common_name}

  FILTER (?location = pl:Garden)
  FILTER (?flower_colour = pl:purple)
  FILTER (?water = pl:None || ?water = pl:Moderate || ?water = pl:Low)
  FILTER (?light = pl:sun)
  FILTER (?soil_requirement = pl:loam)
}
ORDER BY ?plant
```

Figure 12. The query used on the self-made ontology.

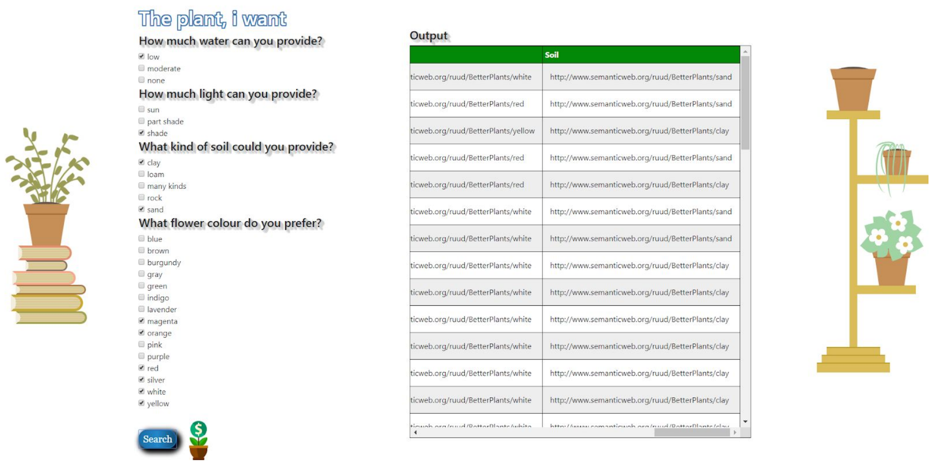
The query that is used on the self made ontology can be seen in figure 12. First, this query gets all the latin names from all the plants that are in the ontology. Then it fetches more data about these plants if they exist, that is why they are being retrieved using the optional command. Finally the data is being filtered, figure 12 is showing a query from an user scenario. In the code the filters are being manipulated, this is visible in figure 13. Not all the data that is retrieved is shown and dynamic in the application. It is chosen to do it that way, because otherwise the application would become too large in size. Also, the user should not be greeted with too many options to choose from because they could be overwhelmed. The application uses the options that are easy to choose from, and the user does not have to have prior knowledge about the subject to choose. The selected options are visible in figure 11.



```
FILTER (?location = pl:Garden) \
FILTER (?flower_colour = pl:" + colourquery + ") \
FILTER (?water = pl:" + waterquery + ") \
FILTER (?light = pl:" + lightquery + ") \
FILTER (?soil_requirement = pl:" + soilquery + ")\
} \
ORDER BY ?plant\
```

Figure 13. How the query is manipulated

Lastly the animated plants were added to the application. This is chosen because a static image would be boring and not inviting to the users, while having too much going on would hurt the users in getting the information about the site, and decrease the user experience. The final animated images do not move too much, the flowers on the right side turn around slowly and the potted plant on the left is also moving slowly. The other plants, books and shelves do not move. The final result of the application can be seen in figure 14.



The plant, i want

How much water can you provide?
[✓ low] moderate none

How much light can you provide?
sun part shade [✓ shade]

What kind of soil could you provide?
[✓ clay] loam many kinds rock [✓ sand]

What flower colour do you prefer?
blue brown burgundy gray green indigo lavender [✓ magenta] [✓ orange] pink purple [✓ red] [✓ silver] [✓ white] [✓ yellow]

Output

	Soil
ticweb.org/ruud/BetterPlants/white	http://www.semanticweb.org/ruud/BetterPlants/sand
ticweb.org/ruud/BetterPlants/red	http://www.semanticweb.org/ruud/BetterPlants/sand
ticweb.org/ruud/BetterPlants/yellow	http://www.semanticweb.org/ruud/BetterPlants/clay
ticweb.org/ruud/BetterPlants/red	http://www.semanticweb.org/ruud/BetterPlants/sand
ticweb.org/ruud/BetterPlants/red	http://www.semanticweb.org/ruud/BetterPlants/clay
ticweb.org/ruud/BetterPlants/white	http://www.semanticweb.org/ruud/BetterPlants/sand
ticweb.org/ruud/BetterPlants/white	http://www.semanticweb.org/ruud/BetterPlants/sand
ticweb.org/ruud/BetterPlants/white	http://www.semanticweb.org/ruud/BetterPlants/clay
ticweb.org/ruud/BetterPlants/white	http://www.semanticweb.org/ruud/BetterPlants/clay
ticweb.org/ruud/BetterPlants/white	http://www.semanticweb.org/ruud/BetterPlants/clay
ticweb.org/ruud/BetterPlants/white	http://www.semanticweb.org/ruud/BetterPlants/clay
ticweb.org/ruud/BetterPlants/white	http://www.semanticweb.org/ruud/BetterPlants/clay
ticweb.org/ruud/BetterPlants/white	http://www.semanticweb.org/ruud/BetterPlants/clay

Figure 14. The application

7 User scenario

An user scenario could be as follows: John works for a large company and they just moved to a new building. The new building has no furniture yet and no accessories, it looks very bland and not inviting to customers. John has been asked by his boss to do something about this, the boss suggests to buy some plants. John is getting stressed by this task because he has no knowledge about plants. When John tells his friends about his problem, someone mentions this website “The plant, i want” and they start looking immediately on the web. John is surprised by the easy user interface and selects some checkboxes to his liking. The application gives quick results and John selects some plants from the output box and purchases these.

8 Conclusion

In the end of the project, a working application was created, that is usable with a running stardog and an internet connection. A user could fill in their needs through a form, by ticking boxes to their liking. This way, the user of the application “The plant, I want” can have a lot of knowledge about plants and select their ideal plant, but the application is also usable for a person without experience with plants. The project thus worked out as planned, everyone who is looking for a plant can use the website to get information about plants that fit their needs.

By doing this project, a lot of the knowledge from the earlier weeks of the course knowledge and data was applied in one big project. Even with the previously acquired knowledge, building the ontology and application was still a challenge. Some things turned out to not be possible the way it was planned, and some ideas had to be simplified to fit the project. Overall, the project was challenging and a good final project to use and practise a lot of knowledge, and with a visible and usable result.

Link to the source files of the application:
https://github.com/4444janus/K-D-final_project