First Report of Semantic Web about

Food

Aspir Ahmet
Tobias Dick
Daniel Egger
Stefan Kaufmann

05.11.2018

Contents

1 Introduction

This Knowledge-Graph based application will serve as a online Recipe-Look-Up service. Either by entering certain ingredients, cook time, an old fashioned search query or all of the previously mentioned, the request will yield according results. A web-overlay ensures the interaction with regular users.

The motivation behind this project has a few points that need to be addressed. The first would be the fact that all the ingredients at home do not get wasted, so before buying new ones it would make sense to look up recipes that would require all the ingredients already in possession. The other point is there to help out busy people or lazy students that do not want to spend too much time on a dish.

The project will be based on various data-sources that are gathered all across the world wide web. These sources are stored locally in order to utilize search queries on them without having the delay when consulting external sources. To be precise the gathered data will be mapped onto a vocabulary that is initially selected from <schema.org>. One aim is to expand the selected vocabulary either by defining new relations by ourselves or using parts of already existing ontologies and merging it together with the already existing one. Once the data is ready it is going to be stored in a TripleStore of choice so valid SPARQL query can give us certain triples. One candidate is Apache Jena Fuseki which is a HTTP based TripleStore that can be accessed by a Java-Client. Once the back-end is ready the goal is to develop a front-end for end-users.

2 Domain Overview

2.1 Description

The topic of this project is simply FOOD. We are focusing on recipes for everyday users which will easy obtain their much desired result. Since there are a lot of properties a recipe can hold we tried to keep it very simple by cutting away things like nutrition. Our ultimate goal is to focus on ingredients and cook-time.

2.2 Data Sources

On the search for data-sources we stumbled upon on a lot very different options. After we got our first data-source we were on the lookout for more sources and found this blog post: click here for the Blog.

The author basically describes different APIs and recipe-based data-sources and gives a short review for each of them. For now are main suppliers are EDAMAM API (1.7+ million recipes), RECIPE-PUPPY API (10.000+ recipes) and a source we found laying on a Amazon-Web-Server (518 recipes).

The data we got was in JSON-format but Edamam also had an own ontology. Since we decided to use <schema.org> as our base we ultimately decided to map every JSON source to JSON-LD.

3 Initial Vocabulary Selection and Domain Specification

3.1 Initial vocabulary

Our first idea was to use schema.org because of its popularity. We were able to find a schema.org class that fits our intentions. The main class is <`https://schema.org/Recipe`> as seen in Figure 1.

Recipe

Canonical URL: http://schema.org/Recipe

Thing > CreativeWork > HowTo > Recipe

A recipe. For dietary restrictions covered by the recipe, a few common restrictions are enumerated via suitableForDiet. The keywords property can also be used to add more detail.

Usage: Between 10 and 100 domains

Figure 1: The Recipe class of schema.org

The properties of this class already covered most of our wanted vocabulary (Figure 2). There were also some properties we used that were not directly related to this class but seemed nonetheless important to ourselves.

Property	Expected Type	Description
Properties from Recipe		
cookTime	Duration	The time it takes to actually cook the dish, in ISO 8601 duration format.
cookingMethod	Text	The method of cooking, such as Frying, Steaming, ...
nutrition	NutritionInformation	Nutrition information about the recipe or menu item.
recipeCategory	Text	The category of the recipe—for example, appetizer, entree, etc.
recipeCuisine	Text	The cuisine of the recipe (for example, French or Ethiopian).
recipeIngredient	Text	A single ingredient used in the recipe, e.g. sugar, flour or garlic. Supersedes ingredients.
recipeInstructions	CreativeWork or ItemList or Text	A step in making the recipe, in the form of a single item (document, video, etc.) or an ordered list with HowToStep and/or HowToSection items.
recipeYield	QuantitativeValue or Text	The quantity produced by the recipe (for example, number of people served, number of servings, etc).
suitableForDiet	RestrictedDiet	Indicates a dietary restriction or guideline for which this recipe or menu item is suitable, e.g. diabetic, halal etc.

Figure 2: Properties of the Recipe class

3.2 Mapping of metadata of the source to the selected vocabularies/ontologies

The Edamam API provides a JSON result in the structure as seen in Figure 3.


```
{
  "q" : "chicken",
  "from" : 0,
  "to" : 1,
  "params" : {
    "sane" : [ ],
    "q" : [ "chicken" ],
    "from" : [ "0" ],
    "app_key" : [ "0a2cfb0cce312b298bf239c7c37790a8" ],
    "to" : [ "1" ],
    "app_id" : [ "6362f010" ]
  },
  "more" : true,
  "count" : 185794,
  "hits" : [ {
    "recipe" : {
      "uri" : "http://www.edamam.com/ontologies/edamam.owl#recipe_7bf4a371c6884d809682a72808da7dc2",
      "label" : "Teriyaki Chicken",
      "image" : "https://www.edamam.com/web-img/262/262b4353ca25074178ead2a07cdf7dc1.jpg",
      "source" : "David Lebovitz",
      "url" : "http://www.davidlebovitz.com/2012/12/chicken-teriyaki-recipe-japanese-farm-food/",
      "shareAs" : "http://www.edamam.com/recipe/teriyaki-chicken-7bf4a371c6884d809682a72808da7dc2/chicken",
      "yield" : 6.0,
      "dietLabels" : [ "Low-Carb" ],
      "healthLabels" : [ "Sugar-Conscious", "Peanut-Free", "Tree-Nut-Free", "Alcohol-Free" ],
      "cautions" : [ ],
      "ingredientLines" : [ "1/2 cup (125ml) mirin", "1/2 cup (125ml) soy sauce", "One 2-inch (5cm) piece of fresh ginger, peeled and grated",
"2-pounds (900g) boneless chicken thighs (4-8 thighs, depending on size)" ],
      "ingredients" : [ {
        "text" : "1/2 cup (125ml) mirin",
        "weight" : 122.99850757795392
      }, {
        "text" : "1/2 cup (125ml) soy sauce",
        "weight" : 134.72774670265568
      }, {
        "text" : "One 2-inch (5cm) piece of fresh ginger, peeled and grated",
        "weight" : 15.0
      }, {
        "text" : "2-pounds (900g) boneless chicken thighs (4-8 thighs, depending on size)",
        "weight" : 907.18474
      } ],
      "calories" : 2253.101981306866,
      "totalWeight" : 1179.9109942806097,
      "totalTime" : 0.0,
      "totalNutrients" : {
        "ENERC_KCAL" : {
          "label" : "Energy",
          "quantity" : 2253.101981306866,
          "unit" : "kcal"
        }
```


Figure 3: A JSON result from the Edamam API

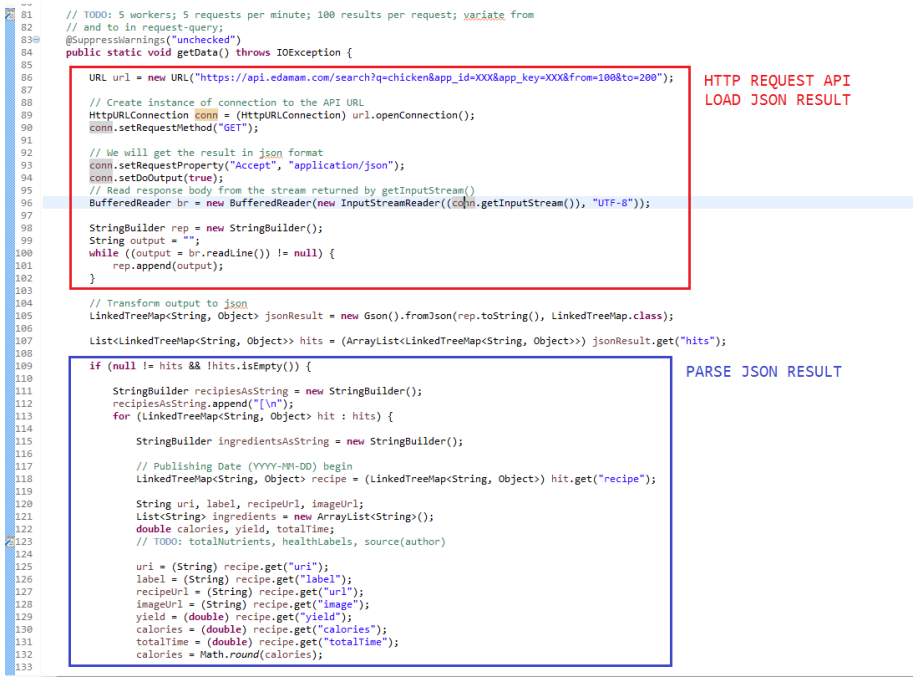
A summary of the mapping:

```
uri -> identifier
label -> name
url -> recipeUrl
image -> image
yield -> recipeYield
calories -> calories
totalTime -> totalTime
ingredientLines -> recipeIngredient
```

3.3 Implementation of mapper tool

Using a java program both the crawler and mapper were implemented. In the snippets below the function call can be seen that sends an HTTP request to the API, reads the JSON result, parses it and maps it to our own vocabulary.

Finally a simple string is generated which represents the JSON-LD file (LINE 151+).



3.4 Implementation of crawler tool

The crawler is basically the previous function call wrapped in to a Callable class in Java so the data gathering can happen simultaneously. One thing is important to keep in mind: the Edamam API restricts the calls per minute and results per request.

With 5 available calls per minute and only 100 results per request we have to implement a tiny waiting routine and a flexible selection of results. This is done by tweaking the parameters <FROM> and <TO> in the API request call.

In the following example 100 and 200: `https://api.edamam.com/search?q=chicken&app_id=XXX&app_key=XXX&from=100&to=200`

```
42
43      public static void crawl() throws InterruptedException, FileNotFoundException, ExecutionException {
44          ExecutorService pool = Executors.newFixedThreadPool(5);
45          ArrayList<Future<String>> results = new ArrayList<Future<String>>();
46          int i = 0;
47          boolean endNotReached = true;
48          while (endNotReached) {
49              results.add(pool.submit(new EdamamCrawler(i, i + 100)));
50              results.add(pool.submit(new EdamamCrawler(i + 100, i + 200)));
51              results.add(pool.submit(new EdamamCrawler(i + 200, i + 300)));
52              results.add(pool.submit(new EdamamCrawler(i + 300, i + 400)));
53              results.add(pool.submit(new EdamamCrawler(i + 400, i + 500)));
54              i += 500;
55              Thread.sleep(1200);
56
57              for (Future<String> res : results) {
58                  try {
59                      if (res.get() == null) {
60                          endNotReached = false;
61                      }
62                  } catch (ExecutionException e) {
63                      // TODO Auto-generated catch block
64                      e.printStackTrace();
65                  }
66              }
67          }
68
69          File recipesFromEdamam = new File("recipesFromEdamam.jsonld");
70
71          PrintWriter tempWriter = new PrintWriter(recipesFromEdamam);
72
73          for (Future<String> res : results) {
74              tempWriter.print(res.get());
75              tempWriter.flush();
76          }
77
78          tempWriter.close();
79
80      }
```

Figure 4: The Crawler Implementation

3.5 Ontology extension

Since the selected properties are not classes themselves we will try to make a class out of recipeIngredient for example. Currently the information for this property is stored as a String but it would make sense to introduce properties like “amount” and “name” and maybe some more attributes for this new class (Figure 5).

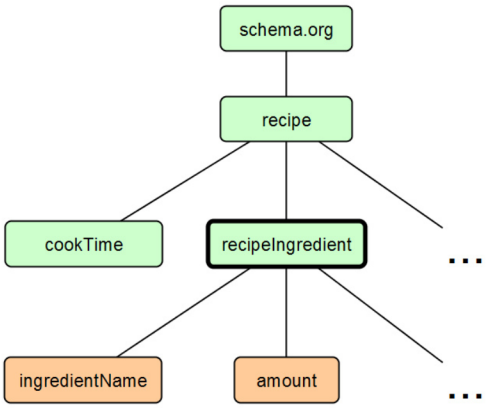


Figure 5: The Described extension-point of the Recipe class

4 Exploratory queries on the loaded dataset

4.1 Small overview of the selected triple store

The selected TripleStore is Apache Jena Fuseki (`https://jena.apache.org/documentation/fuseki2/`).

We are using the HTTP TripleStore from Jena, because its easy to handle from a Java client. For now Fuseki is launched from a separate path and runs on localhost on port 3030 and provides a GUI via web (Figure 6).

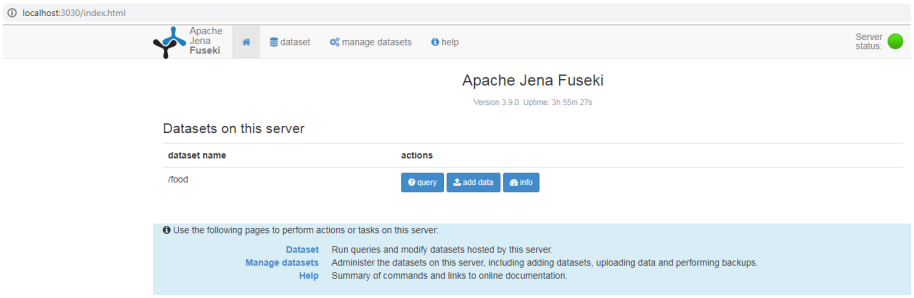


Figure 6: The Apache Jena Fuseki GUI

Pretty much everything can be done with the Java client if the permissions are granted. Fuseki usually has endpoints that can be accessed for certain operations (Figure 7).

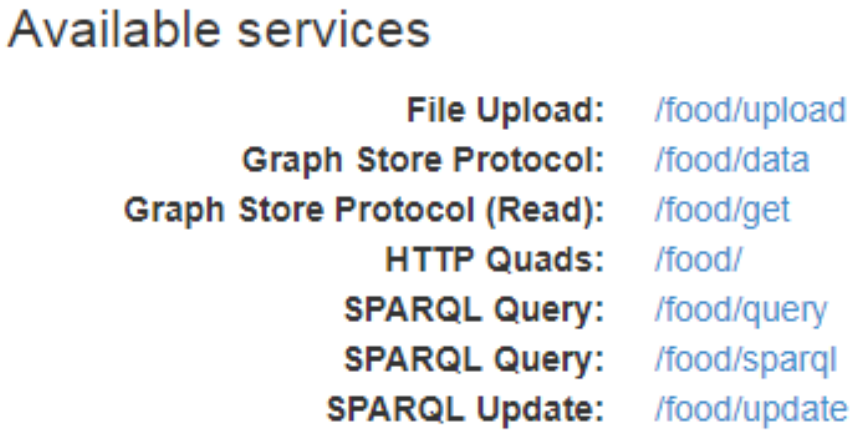


Figure 7: The Apache Jena Fuseki services

Once graphs are loaded they can be stored on hard disk or the main memory depending on the purpose. For our purposes we kept our graphs on the hard disk. The data-set that's on the fuseki is easily accessed by local methods can be seen in Figure 8 and Figure 9 below.

```
public void deleteModel(String graphName) {
    DatasetAccessorFactory.createHTTP(connectionUrl + "/" + dataName + "/data").deleteModel(graphName);
}

public void deleteDefaultModel() {
    DatasetAccessorFactory.createHTTP(connectionUrl + "/" + dataName + "/data").deleteDefault();
}

public void addModel(String graphName, Model model) {
    DatasetAccessorFactory.createHTTP(connectionUrl + "/" + dataName + "/data").add(graphName, model);
}

public void addDefaultModel(Model model) {
    DatasetAccessorFactory.createHTTP(connectionUrl + "/" + dataName + "/data").add(model);
}
```

Figure 8: Load and delete operations on desired graphs

The SPARQL query is deployed in a method which requires the query itself as a string.

```
public void query(String queryString) {

    Query query = QueryFactory.create(queryString);

    RDFConnection queryConnection = RDFConnectionRemote.create().destination(connectionUrl + "/")
            .queryEndpoint(dataName + "/sparql")
            .acceptHeaderSelectQuery("application/sparql-results+json, application/sparql-results+xml;q=0.9")
            .build();
    try (RDFConnection conn = queryConnection) {
        conn.queryResultSet(query, ResultSetFormatter::out);

    }

}
```

Figure 9: Deploying a SPARQL query

4.2 Example query

We want to have a small section were we explain the thought process behind our queries. The general setup are named graphs for data sources and ontologies. Union of all named graphs is currently the default graph (not really needed since we are very flexible with our queries).

Example query with detailed explanation: Get all classes from data-set (data-set is a single movie entry).

```
SELECT DISTINCT ?ox WHERE{
  { GRAPH ?g1{?s ?p ?o . ?s a schema:Movie . ?s ?px ?prop . ?prop a ?ox} .
    GRAPH ?g2 {?ox a rdfs:Class}
  } UNION
  { GRAPH ?g3{?sy ?py ?ox . ?sy a schema:Movie }.
    GRAPH ?g4 {?ox a rdfs:Class}
  }
}
```

The query above is the solution for this and is explained in each step.

The GRAPH keyword goes trough the named graphs and binds one of them to the subset within the parentheses. That means if the ontologies need to be accessed this has to be done outside with a separate GRAPH call.

The key idea behind this query is to first get each subjects that have the rdf:type schema:Movie.

```
?s ?p ?o . ?s a schema:Movie

_:Ndd0b1c9963c94e488a94ba4eef20b35e <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Movie> .
```

This way we can be sure that our GRAPH is bound to the named graph which contains the data. From there we have to find each objects where a reference from the main subject can be back traced. We assume that from the main subject there are certain predicates that address properties.

```
?s ?px ?prop

_:Ndd0b1c9963c94e488a94ba4eef20b35e <http://schema.org/description> "Jack Sparrow ... his daughter are after it too." .
_:Ndd0b1c9963c94e488a94ba4eef20b35e <http://schema.org/actor> _:N953c7c07506e47089525b55b681aac27 .
_:Ndd0b1c9963c94e488a94ba4eef20b35e <http://schema.org/author> _:N2cf805bd409f480e9b35426e383056b8 .
_:Ndd0b1c9963c94e488a94ba4eef20b35e <http://schema.org/actor> _:Nc6f0ac13f81f47e58507f27d3a9d6494 .
_:Ndd0b1c9963c94e488a94ba4eef20b35e <http://schema.org/aggregateRating> _:N1490725ea37a453986858132310a882b .
_:Ndd0b1c9963c94e488a94ba4eef20b35e <http://schema.org/author> _:Na95e065f322e4103bb3520ed73326ab4 .
_:Ndd0b1c9963c94e488a94ba4eef20b35e <http://schema.org/director> _:N5848fd45bb3b4cec8fc179cdf166829c .
_:Ndd0b1c9963c94e488a94ba4eef20b35e <http://schema.org/actor> _:N97491262314543ef8711acbd3a0a3306 .
_:Ndd0b1c9963c94e488a94ba4eef20b35e <http://schema.org/name> "Pirates of the Carribean: On Stranger Tides (2011)" .
```

Next up we want to determine the type of the objects ?prop that are mentioned in the main movie subject and get all ?ox.

```
?prop a ?ox

_:N2cf805bd409f480e9b35426e383056b8 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Person> .
_:Ndd0b1c9963c94e488a94ba4eef20b35e <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Movie> .
_:N953c7c07506e47089525b55b681aac27 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Person> .
_:N1490725ea37a453986858132310a882b <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/AggregateRating> .
_:Na95e065f322e4103bb3520ed73326ab4 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Person> .
_:Nc6f0ac13f81f47e58507f27d3a9d6494 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Person> .
_:N97491262314543ef8711acbd3a0a3306 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Person> .
_:N5848fd45bb3b4cec8fc179cdf166829c <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Person> .
```

Once we get the needed results from the dataset its up to the second named graph which is the ontology to find out if the results are rdfs:Class types. For the example we just show it for schema:Person.

```
GRAPH ?g2 {?ox a rdfs:Class}

<http://schema.org/Person> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2000/01/rdf-schema#Class> .
```

The second part of the query is to ensure we get the movie class itself as well. This is done by utilizing the keyword UNION. Union means there are two sets to select the results from. If only one column is desired keep the variable name same, otherwise write the variable names of each subset in the SELECT.

```
17
18 SELECT DISTINCT ?ox WHERE{
19   { GRAPH ?g1{?s ?p ?o . ?s a schema:Movie . ?s ?px ?prop . ?prop a ?ox} .
20     GRAPH ?g2 {?ox a rdfs:Class}
21   } UNION
22   { GRAPH ?g3{?sy ?py ?ox . ?sy a schema:Movie }.
23     GRAPH ?g4 {?ox a rdfs:Class}
24   }
25 }
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 3 of 3 entries

Search: Show 50 entries

	ox
1	schema:Person
2	schema:AggregateRating
3	schema:Movie

4.3 Results of exploratory queries

In this section we want to show the constructed SPARQL-queries including their results.

4.3.1 Total number of triples

```
SELECT (COUNT(?x) as ?triples) WHERE
{
  ?x ?y ?s . ?x a schema:Recipe
}
```

```
numberOfTriples
-----------
| triples |
===========
| 11454   |
-----------
```

4.3.2 Total number of instantiations

```
SELECT  ?class (COUNT(?x) as ?instances) WHERE
{
  ?x ?y ?class . ?class a rdfs:Class .
  ?x a schema:Recipe
} GROUP BY ?class
```

```
numberOfTriplesPerClass
-----------------------------------------
| class                       | instances |
=========================================
| <http://schema.org/Recipe> | 618       |
-----------------------------------------
```

4.3.3 Total number of distinct classes

```
SELECT (COUNT(*) as ?numberOfDistinctProperties) WHERE
{ SELECT DISTINCT ?Properties WHERE
  {
    ?x ?Properties ?z. ?Properties a rdf:Property .
    ?x a schema:Recipe
  } GROUP BY ?Properties
}
```

```
numberOfDistinctClasses
---------------------------
| numberOfDistinctClasses |
===========================
| 1                       |
---------------------------
```

4.3.4 Total number of distinct properties

```
SELECT (COUNT(*) as ?numberOfDistinctProperties) WHERE
{
  SELECT DISTINCT ?Properties WHERE
  {
    ?x ?Properties ?z. ?Properties a rdf:Property . ?x a schema:Recipe
  } GROUP BY ?Properties
}
```

```
-----------------------------
| numberOfDistinctProperties |
=============================
| 12                        |
-----------------------------
```

4.3.5 List of all classes used in dataset per data source

```
SELECT DISTINCT ?graph ?class WHERE
{
    GRAPH ?graph { ?s ?p ?class . ?s a schema:Recipe}  .
    GRAPH ?j {?class a rdfs:Class}
}
```

```
classesPerDataSet
-------------------------------------------------------------------------
| graph                                  | class                        |
=========================================================================
| <http://localhost:3030/food/data/google> | <http://schema.org/Recipe> |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/Recipe> |
-------------------------------------------------------------------------
```

4.3.6 List of all properties used in dataset per data source

```
SELECT DISTINCT ?namedGraph ?class
{
    GRAPH ?namedGraph { ?s ?class ?o  . ?s a schema:Recipe }.
    ?class a rdf:Property
} GROUP BY ?class ?namedGraph ORDER BY ?namedGraph
```

```
propertiesPerDataSet
-----------------------------------------------------------------------------------------------
| namedGraph                             | class                                              |
===============================================================================================
| <http://localhost:3030/food/data/edamam> | <http://schema.org/author>                       |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/identifier>                   |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/image>                        |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/name>                         |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/nutrition>                    |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/recipeIngredient>             |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/recipeYield>                  |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/totalTime>                    |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/url>                          |
| <http://localhost:3030/food/data/edamam> | <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> |
| <http://localhost:3030/food/data/google> | <http://schema.org/author>                       |
| <http://localhost:3030/food/data/google> | <http://schema.org/cookTime>                     |
| <http://localhost:3030/food/data/google> | <http://schema.org/image>                        |
| <http://localhost:3030/food/data/google> | <http://schema.org/name>                         |
| <http://localhost:3030/food/data/google> | <http://schema.org/prepTime>                     |
| <http://localhost:3030/food/data/google> | <http://schema.org/recipeIngredient>             |
| <http://localhost:3030/food/data/google> | <http://schema.org/recipeYield>                  |
| <http://localhost:3030/food/data/google> | <http://schema.org/url>                          |
| <http://localhost:3030/food/data/google> | <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> |
-----------------------------------------------------------------------------------------------
```

4.3.7 Total number of instances per class per data source

```
SELECT DISTINCT ?namedGraph ?class (COUNT(?class) as ?instances)
{
 GRAPH ?namedGraph { ?s ?p ?class . ?s a schema:Recipe}  . ?class a rdfs:Class
} GROUP BY ?class ?namedGraph ORDER BY ?namedGraph
```

```
instancesPerClassPerDataSet
-------------------------------------------------------------------------------------
| namedGraph                                | class                      | instances |
=====================================================================================
| <http://localhost:3030/food/data/edamam> | <http://schema.org/Recipe> | 100       |
| <http://localhost:3030/food/data/google> | <http://schema.org/Recipe> | 518       |
-------------------------------------------------------------------------------------
```

4.3.8 Total number of distinct subjects per property per data source

```
SELECT ?namedGraph ?class (COUNT(?subjectCount) as ?subjects) WHERE
{
    SELECT ?namedGraph ?class (COUNT(?s) as ?subjectCount)
    {
        GRAPH ?namedGraph { ?s ?class ?o . ?s a schema:Recipe } . ?class a rdf:Property
    } GROUP BY ?s ?class ?namedGraph ORDER BY ?namedGraph
} GROUP BY ?s ?class ?namedGraph ORDER BY ?namedGraph
```

```
subjectsPerPropertyPerDataSet
-----------------------------------------------------------------------------------------------------------------
| namedGraph                                | class                                                 | subjects |
=================================================================================================================
| <http://localhost:3030/food/data/edamam> | <http://schema.org/author>                            | 100      |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/identifier>                        | 100      |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/image>                             | 100      |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/name>                              | 100      |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/nutrition>                         | 100      |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/recipeIngredient>                  | 100      |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/recipeYield>                       | 100      |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/totalTime>                         | 100      |
| <http://localhost:3030/food/data/edamam> | <http://schema.org/url>                               | 100      |
| <http://localhost:3030/food/data/edamam> | <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>     | 100      |
| <http://localhost:3030/food/data/google> | <http://schema.org/author>                            | 518      |
| <http://localhost:3030/food/data/google> | <http://schema.org/cookTime>                          | 518      |
| <http://localhost:3030/food/data/google> | <http://schema.org/image>                             | 518      |
| <http://localhost:3030/food/data/google> | <http://schema.org/name>                              | 518      |
| <http://localhost:3030/food/data/google> | <http://schema.org/prepTime>                          | 518      |
| <http://localhost:3030/food/data/google> | <http://schema.org/recipeIngredient>                  | 518      |
| <http://localhost:3030/food/data/google> | <http://schema.org/recipeYield>                       | 518      |
| <http://localhost:3030/food/data/google> | <http://schema.org/url>                               | 518      |
| <http://localhost:3030/food/data/google> | <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>     | 518      |
-----------------------------------------------------------------------------------------------------------------
```

4.3.9 Total number of distinct objects per property per data source

```
SELECT ?namedGraph ?class (COUNT(?subjectCount) as ?objects) WHERE{
  SELECT ?namedGraph ?class (COUNT(?o) as ?subjectCount)
    { GRAPH ?namedGraph { ?s ?class ?o . ?s a schema:Recipe  }
    . ?class a rdf:Property
    } GROUP BY ?o ?class ?namedGraph ORDER BY ?namedGraph
} GROUP BY ?o ?class ?namedGraph ORDER BY ?namedGraph
```

objectsPerPropertyPerDataSet

namedGraph	class	objects
<http://localhost:3030/food/data/edamam>	<http://schema.org/author>	1
<http://localhost:3030/food/data/edamam>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	1
<http://localhost:3030/food/data/edamam>	<http://schema.org/identifier>	100
<http://localhost:3030/food/data/edamam>	<http://schema.org/image>	100
<http://localhost:3030/food/data/edamam>	<http://schema.org/nutrition>	100
<http://localhost:3030/food/data/edamam>	<http://schema.org/url>	100
<http://localhost:3030/food/data/edamam>	<http://schema.org/recipeYield>	12
<http://localhost:3030/food/data/edamam>	<http://schema.org/totalTime>	32
<http://localhost:3030/food/data/edamam>	<http://schema.org/recipeIngredient>	672
<http://localhost:3030/food/data/edamam>	<http://schema.org/name>	87
<http://localhost:3030/food/data/google>	<http://schema.org/author>	1
<http://localhost:3030/food/data/google>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	1
<http://localhost:3030/food/data/google>	<http://schema.org/prepTime>	28
<http://localhost:3030/food/data/google>	<http://schema.org/recipeIngredient>	3631
<http://localhost:3030/food/data/google>	<http://schema.org/cookTime>	37
<http://localhost:3030/food/data/google>	<http://schema.org/name>	516
<http://localhost:3030/food/data/google>	<http://schema.org/image>	517
<http://localhost:3030/food/data/google>	<http://schema.org/url>	518
<http://localhost:3030/food/data/google>	<http://schema.org/recipeYield>	9

4.3.10 Distinct properties used on top 5 classes

```
SELECT DISTINCT ?properties  WHERE {
  ?properties schema:domainIncludes ?o .
  ?sub ?properties ?obj .
  FILTER(?o = ?class) {
    SELECT ?class  WHERE {
        ?properties ?p ?class .
        ?class a rdfs:Class .
        ?properties a ?classtype .
        FILTER (?classtype IN (schema:Recipe))
      } GROUP BY ?class ORDER BY DESC(?instances) LIMIT 5
    }
}
```

propertiesInTop5Classes

properties
<http://schema.org/recipeIngredient>
<http://schema.org/recipeYield>
<http://schema.org/cookTime>
<http://schema.org/nutrition>

4.3.11 Distinct wikidata types that may be aligned with

```
SELECT DISTINCT ?item ?itemLabel ?localClass WHERE {
   {SELECT DISTINCT ?localClass ?z WHERE {
      ?s ?p ?localClass . ?localClass a rdfs:Class . ?s a schema:Recipe . ?localClass rdfs:label ?z .}}
      SERVICE <https://query.wikidata.org/sparql> {
        SERVICE wikibase:mwapi {
        bd:serviceParam wikibase:api "EntitySearch" .
        bd:serviceParam wikibase:endpoint "www.wikidata.org" .
        bd:serviceParam mwapi:search ?z .
        bd:serviceParam mwapi:language "en" .
        bd:serviceParam mwapi:limit 5 .
        ?item wikibase:apiOutputItem mwapi:item .
      } ?item rdfs:label ?itemLabel .
      FILTER(LANG(?itemLabel) = "" || LANGMATCHES(LANG(?itemLabel), "en"))      }
}
```

wikiDataAlignment

item	itemLabel	localClass
<http://www.wikidata.org/entity/Q219239>	"recipe"@en	<http://schema.org/Recipe>
<http://www.wikidata.org/entity/Q605076>	"cookbook"@en	<http://schema.org/Recipe>
<http://www.wikidata.org/entity/Q605076>	"Cookbook"@en-ca	<http://schema.org/Recipe>
<http://www.wikidata.org/entity/Q605076>	"cookery book"@en-gb	<http://schema.org/Recipe>
<http://www.wikidata.org/entity/Q7759779>	"The Recipe"@en	<http://schema.org/Recipe>
<http://www.wikidata.org/entity/Q11241662>	"Recipe"@en	<http://schema.org/Recipe>
<http://www.wikidata.org/entity/Q21188738>	"Recipe"@en	<http://schema.org/Recipe>

5 Conclusion and Next Steps

The application is up and running and is available to give back results for certain criteria when asked. The aim was to have a functioning TripleStore with data that represents an ontology so semantics can be applied to raw data. Querying the data via SPARQL and a Java client also works well and will most definitely not hinder us on further progress. A milestone was to get the requested queries to run which we accomplished. What we need to focus on is to extend the current vocabulary we are using and also think of a possible framework for the frontend.