8-9-2014

VOCAB4ME: A TOOL THAT PROVIDES VOCABULARY RECOMMENDATIONS FOR PUBLISHING LINKED DATA

Srikar Nadipally
University of South Carolina - Columbia

VOCAB4ME: A TOOL THAT PROVIDES VOCABULARY RECOMMENDATIONS FOR PUBLISHING LINKED DATA

by

Srikar Nadipally

Bachelor of Technology
Kakatiya University, 2009

Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Science in

Computer Science & Engineering

College of Engineering & Computing

University of South Carolina

2014

Accepted by:

Manton M. Matthews, Director of Thesis

Marco Valtorta, Reader

Colin F. Wilder, Reader

Lacy Ford, Vice Provost and Dean of Graduate Studies

DEDICATION

To my parents... who were always there with me!

To my sister... who is my biggest supporter!

To my friends... who were always there by me!

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Manton M. Matthews, for giving me an excellent opportunity to work under his supervision on this thesis.

I would also like to thank Dr. Colin F. Wilder, Associate Director Center for Digital Humanities, for giving me an amazing opportunity to work as a Research Assistant. My work at CDH has helped me immensely in doing this thesis.

Last, but not definitely the least, I would like to thank Dr. Marco Valtorta for being on my thesis committee.

ABSTRACT

The web before linked data was a database of html documents. These documents were meant for human consumption and it was hard for machines to make sense of data in html documents. The linked data was introduced with the aim of making the web a global database of data that is machine processable. Linked Data describes a method of publishing structured data so that it can be interlinked and become more useful. Realizing the promise of linked data a lot of people started publishing linked data. But the process of publishing the huge amount of existing data is cumbersome and usually takes someone very knowledgeable to do it. Publishing linked data on the web requires finding appropriate vocabularies that describe the semantics of the data. Finding such vocabularies is difficult to a new user. The proposed system will suggest vocabularies to use when somebody is trying to publish linked data. The system does so by using string similarity metrics to match entity and property names in our dataset to Class and Property names in existing RDF vocabularies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

HTML..... Hyper Text Markup Language

LOD.....Linked Open Data

OWL..... Web Ontology Language

RDF..... Resource Description Framework

RDFS .....RDF Schema

URI .....Uniform Resource Identifier

URL .....Uniform Resource Locator

URN ..... Uniform Resource Name

XML..... Extensible Markup Language

CHAPTER 1

BACKGROUND

The web has the huge amount of information about topics varying from productively performing daily tasks to information on cutting edge research. But this information is not in a structured format so we are not able to achieve the full potential of the web. The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation[4]. To understand why we need semantic web, we need to understand what is classic web and what are its shortcomings.

1.1 CLASSIC WEB & ITS SHORTCOMINGS:

The classic web i.e., the web before semantic web was envisioned for human consumption. The classic web is made up of HTML documents, these documents are good for humans because humans can read and understand the information contained in the html document. HTML documents have URIs and can be uniquely identified and linked to from any document on the web. For this reason, we call the classic web as a web of documents. Figure 1.1 shows the architecture of classic Web.

Because the classic web was envisioned with human consumption in mind, the degree of structure in the document is fairly low. The contents of the semantics of the document are implicit and it is hard for the machine to tell what a page is talking about. Currently search engines are one of the major consumers of data and because of the lack of structure in the documents, they cannot give us what we search for. Consider the following example that is taken from http://www.mpi-inf.mpg.de/yago-naga/CIKM10-tutorial/, if we wanted to know if there will ever be another like Elvis Presley and search Google for "Another Elvis", the results we get back are related to Elvis Presley. If we search for "Another singer called Elvis, young" the results we still get are about Elvis Presley. The classic web also does not allow us to ask expressive queries like "give me all Soccer players with tricot number 11, playing for a club having a stadium with over 40,000 seats and is born in a country with over 10 million inhabitants?"

Classic web also fails to answer complex queries involving background knowledge like find information about "animals that use sonar but are not either bats or dolphins".

One other major problem with the current web is that it does not define any standard structure for data, as a result it is very difficult to combine data from multiple sources.

1.2 ALTERNATIVE 1 WEB API'S & MASHUPS:

A Web API is a programmatic interface to expose data in a structured format. Mashups are web applications that combine the use of multiple Web API's. One example of a Mashup would be DogPile, which is a search engine that returns the search results by combining the search results from other leading search engines. Any application developer can study the documentation of the Web API or Mashups and write programs that understand the semantics of the data.

Even though Web APIs and Mashups provide data in a structured format, their use is very limited mainly because they amount of data they expose is very very small compared to the vast amount of web sites which do not expose data using web services. Also there is no provision to link this data to other related data and so it can help in discovery of other related data.

1.3 ALTERNATIVE 2 MICROFORMATS:

Microformats embed structured data into HTML pages, by adding attributes to existing HTML elements, browsers ignore these attributes, but semantic applications can use these attribute to infer the structure and semantics of the data. Examples of Microformats include hCard, hCalendar, hReview e.t.c. Figure 1.3 is an example of an event that is described using Microformats.

Although the number is increasing, currently there is only fixed set of Microformats that we can use. Just like Web API's and mashups, there is no provision to link data to other related data.

1.4 ALTERNATIVE 3 SEMANTIC WEB:

Since the existing alternatives are not sufficient, semantic web was proposed. Now that we understand why we need semantic web and related technologies, lets look at what is semantic web and what are the various semantic web technologies.

Semantic Web: Tim Berners Lee et al coined the term Semantic Web classic paper on Semantic web. They defined Semantic Web as an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.

There are many definitions for semantic web the easiest one was defined by Bob Du Charme, the author of the book Learning SPARQL. According to him semantic web is "*a set of standards and best practices for sharing data and the semantics of that data over the Web for use by applications*".

Semantic web introduced standards like RDF, RDFS/OWL, SPARQL. RDF is meant to be used to share data, RDFS and OWL are used to share the semantics of the data. SPARQL is used for querying the data.

Resource Description Framework (RDF): RDF allows us to make statements about resources and the relationship between the resources. A resource can be any thing or a concept in the world. It can be a book or a movie or a person, It can also be an abstract thing like a disease. RDF allows us to make statements about all these types of things. One of the key requirements of a resource is that it

should be uniquely identifiable by a universally unique name and we generally use Uniform Resource Identifiers for this purpose.

An RDF statement is of the form *subject predicate object*. Because an rdf statement is made up of three elements, we call RDF statements as triples. The subject and predicate of a triple should always be resources i.e. uniquely identifiable using a URI. The Object can be either a resource or a literal. The following is an example of an informal RDF statement. In the actual statement we will be having URIs instead of just the names, but for the sake of brevity lets consider the informal triple.

<shakespeare> <authorOf> <hamlet>.

When a human reads this and if he knows about Shakespeare, he can infer that Shakespeare who is a person is the author of the book hamlet. But a machine does not know what any of the terms Shakespeare, authorOf or hamlet means. This information has to be supplied explicitly. Let us expand our RDF knowledge so that a machine can understand.

<shakespeare> <type> <Person>

<hamlet> <type> <Book>

<shakespeare> <hasName> "William Shakespeare"

<hamlet> <hasName> "The Tragedy of Hamlet, Prince of Denmark"

<shakespeare> <authorOf> <hamlet>.

Now that we expanded our triples, a machine can understand what the triple means.

RDF as a Graph: An RDF document is best visualized as a graph. An RDF graph is made up of an ellipse to denote a resource, rectangle or a rounded rectangle is used to denote a literal and properties are denoted using a labeled arrow. The RDF graph for the document described above is shown in figure 1.4.

RDF graphs can be merged very easily and new information can be discovered. If we have a RDF document with a triple that Shakespeare was born in England and we have another RDF document about England, we can merge these two documents and infer new information. An example of merging RDF graphs is shown in figures 1.5 and 1.6.

RDF Serialization: RDF is a data model, if we were to exchange data using this data model, we need some concrete syntactic representation. Saving the RDF data to a file is called RDF serialization. There are a lot of RDF serialization formats.

1. Turtle: a human friendly compact format
2. N-Triples: similar to turtle, but not compact
3. N-Quads: a superset of N-Triples
4. Notation 3(N3): similar to Turtle, with ability to include inference rules
5. RDF/XML: XML based syntax

The RDF serializations in various formats for the document described in the previous section are shown below.

N-Triple Notation:

<*http://example.org/example#shakespeare*>

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

<*http://example.org/example#Person*>.

<*http://example.org/example#shakespeare*>

<*http://example.org/example#hasName*>

"*William Shakespeare*".

RDF/XML Notation:

```
<?xml version="1.0"?>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:example="http://example.org/example#">
      <example:Person rdf:about="http://example.org/example#shakespeare">
      <example:hasName>William Shakespeare</example:hasName>
  </example:Person>
</rdf:RDF>
```

Turtle Notation:

@prefix example: <http://example.org/example#>.

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

example:shakespeare (a/rdf:type) example:Person ;

example:hasName "William Shakespeare";

example:authorOf example:hamlet.

Limitations of RDF: RDF can only make statements about resources, if we want to provide semantics of resources we have to use RDF in conjunction with vocabularies/ontologies. Vocabularies can be defined using RDFS and Ontologies can be defined using OWL. RDF by it self cannot prevent us from making statements like the following.

example:dog authorOf "book".

RDF Schema (RDFS): RDF Schema defines the basic vocabulary that can be used in RDF document to describe resources. RDF Schema is a vocabulary description language, which is built on top of RDF. RDF is very flexible, in that it can only contain triples. It can convey information but it cannot be used to infer things from our triples. RDFS primarily helps us to group things or resources in to classes or hierarchical structures. RDFS is somewhat similar to object oriented systems. It adds the following terms to make it object oriented.

- RDF:type: This can be used to specify that a resource is an instance of a particular class. E.g: "example:Person1 rdf:type example:Person"

- RDF:class, RDFS:Property: These are used to create new classes and properties

    e.g.: example:Person rdf:type rdf:class

    example:firstName rdf:type rdfs:Property

- RDFS:Domain, RDFS:Range: These are used to specify the domain and range of a property.

    e.g.: example:firstName rdfs:domain example:Person

    example:firstName rdfs:range XSD:String

- RDFS:SubClassOf, RDFS:SubPropertyOf: These are used to create hierarchies of classes and properties.

    e.g.: example:Person rdfs:subclassof rdfs:Human

    example:mother rdfs:subpropertyof example:parent

- RDFS:Label – A string of text describing the resource
- RDFS:Comment – A potentially longer comment about the resource
- RDFS:SeeAlso – Links to other "relevant" resources
- RDFS:Literal – Something that is a primitive data type

Apart from these terms, there are other terms defined that help us define things like collections, reification. RDFS allows us to infer new triples from existing triples using entailments. RDFS allows two types of entailments.

Class Entailments:

<Animals *rdfs:subClassof* LivingBeings>

<Cats *rdfs:subClassOf* Animals>

=>

<Cats *rdfs:subClassOf* LivingBeings>

Property Entailments:

<ParentOf *rdfs:subPropertyOf* AncestorOf>

<Fatherof *rdfs:subPropertyOf* ParentOf>

=>

<FatherOf *rdfs:subPropertyOf* AncestorOf>

Limitations of RDFS: RDFS too weak to describe resources in sufficient detail. We cannot specify localized domain and range constraints. For example we cannot say that the range of hasChild is person when applied to persons and elephant when applied to elephants.

It does not have any support for specifying existential or cardinality constraints. For example we cannot say things like a person can have only two parents.

It also does not have provisions for specifying transitive, inverse and symmetrical properties. For example if we have a triple <john> <spouseOf> <jane>. RDFS does not have a way to specify that spouseOf is a symmetrical property, using which we can infer that <jane> <spouseOf> <john>.

Web Ontology Language (OWL): If we need strict semantics for our data, we should define Ontology. Ontologies are defined using OWL. Web ontology language is more expressive than RDFS. It provides all the semantics that RDFS can express but it adds some more terms to be more expressive. There are three flavors of OWL Lite, Full and DL with varying levels of expressivity and restrictions.

In OWL classes can be defined in many ways by combining different classes like Union, Intersection, Enumeration, Restriction and Complement. In OWL we can specify that a person can have at most 2 parents.

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:maxCardinality>2</owl:maxCardinality>
 </owl:Restriction>
```

OWL supports defining symmetric, inverse and transitive properties. We can define a symmetric property called friendOf as shown below.

```
<owl:SymmetricProperty rdf:ID="friendOf">
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range  rdf:resource="#Human"/>
 </owl:SymmetricProperty>
```

Linked Data: The concept of linked data is newer than semantic web itself. It was proposed to make semantic web more useful. It is a method to create and publish structured data that can be interlinked with other structured data, so that the

data becomes more useful. Data that is published as linked data is machine processable.

Linked Data vs. Semantic Web: There are no agreed upon differences or similarities, but the general consensus is that semantic web is made up of linked data. To elaborate Semantic Web is the whole and Linked Data is the parts. Tim Berners-Lee, inventor of the Web and the person credited with coining the terms Semantic Web and Linked Data has frequently described Linked Data as "the Semantic Web done right" Linked Data is newer than the Semantic Web.

Semantic Web Architecture: Semantic web architecture is an extension of the normal web architecture. The bottom layer is made of Unicode character set and URI's; this layer is useful for communication. The next layer is the XML, namespace and schema layer, this is an open standard to share data. As the figure 1.7 shows semantic web adds a few layers on top of existing web standards. RDF Schema, ontologies and logic will be discussed in detail in the following sections. The layers above the logic layer are unrealized i.e. they are just ideas and do not have any standards to implement them yet. We discussed about RDF, RDFS/OWL layers before, we will discuss remaining below.

Logic layer: Provides a universal language for monotone logic, any existing system can export the rules but generally cannot import them. Many inference engines exist which can reason up on the data when provided with additional information in the form of a vocabulary. Inference on the Semantic Web is one of the tools of choice to improve the quality of data integration on the Web, by

discovering new relationships, automatically analyzing the content of the data, or managing knowledge on the Web in general. Inference based techniques are also important in discovering possible inconsistencies in the (integrated) data.

For example, if a data set includes a relation *Flipper isA Dolphin* and Ontology declares that "every Dolphin is also a Mammal", then the inference engine can add the statement that *Flipper is a Mammal* to the set of relationships even though that was not part of the original data.

1.5 PRINCIPLES OF LINKED DATA

Tim Berners Lee outlined the following principles for linked data.

- Use URIs as names of Things as opposed to just documents
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
- Include links to other URIs, so that they can discover more things.

1.6 WHY PUBLISH LINKED DATA

Publishing data in the linked data format has the following advantages.

- Ease of Discovery: Data can be easily discovered, because it is linked to other linked data.
- Ease of Consumption: Data is readily available in machine processable format, which will increase the ease of consumption.
- Reduced Redundancy: Maintaining a single authoritative copy of the data and linking to that data wherever necessary will reduce the redundancy, it will also avoid data duplication and data

inconsistencies that occur because of data duplication whenever the data is one copy of the data is updated.

- Added Value: Eco systems can be built around the data

1.7 TOOLS FOR PUBLISHING LINKED DATA:

The Linked Data community, to help with the creation, linking, testing and publishing of Linked Data, has developed several tools.

Vocabulary/Ontology Creation Tools: If a data publisher cannot find a relevant vocabulary, or existing vocabularies are not good enough/suitable for the use case, they can make their own ontology. Some of the tools that help with ontology/vocabulary creation are Protege, Neologism, Neon Toolkit and SWOOP.

Link Discovery Tools: Simply creating Linked Data that is not linked to other data sources is not enough. The tools like SILK and LIMES can be used to create links between datasets.

Relational Database Mapping Tools: Most of the data is already present in the Relational Databases, which have some sort of structure. If we would like to publish this data as Linked Data, the tools like D2R and Open Link Virtuoso can be used.

Linked Data Validation Tools: Our Linked Data server should serve HTML when the user-agent requests a HTML representation and server other RDF serialization as requested, there are tools like "Vapour Linked Data Validator", "RDF Alerts" and "Sindice Inspector" which help us to test this functionality.

Application Frameworks: Finally there are frameworks like Apache Jena, Sesame that help us in building Semantic Web and Linked Data applications.

1.8 PUBLISHING LINKED DATA THE PROCESS: The process of publishing linked data, as outlined by Chris Bizer involves the following steps.

- Understand your Data
    - What are the main entities in the dataset?
    - What properties does each entity have?
    - What are the relationships between each entity?
- Publish it on the Web as RDF.
    - Select Vocabularies
    - Partition the RDF graph in to data pages
    - Assign URI to each data page
    - Create HTML Variants of each data page
    - Assign URI to each entity
    - Add page metadata and link sugar
    - Add a semantic sitemap.
- Link it with other Data Sources.

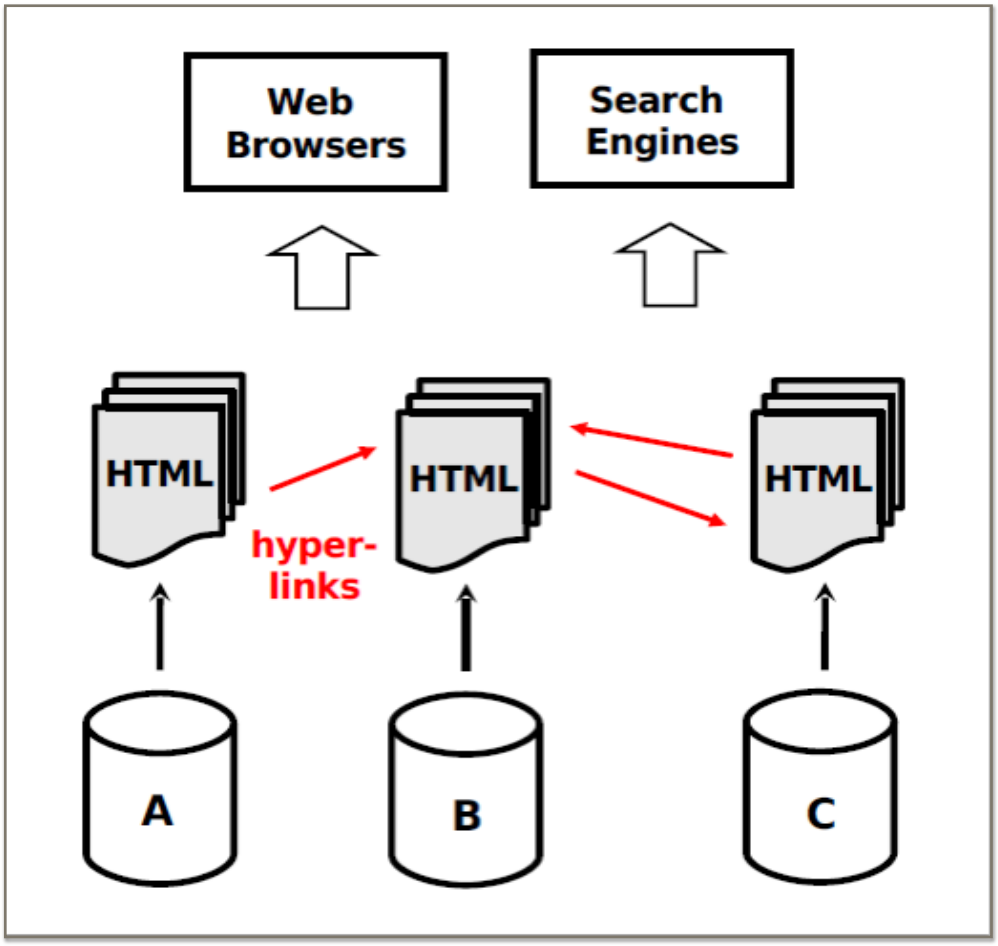


Figure 1.1 Classic Web
(From http://events.linkeddata.org/iswc2008tutorial/how-to-publish-linked-data-iswc2008-slides.pdf)

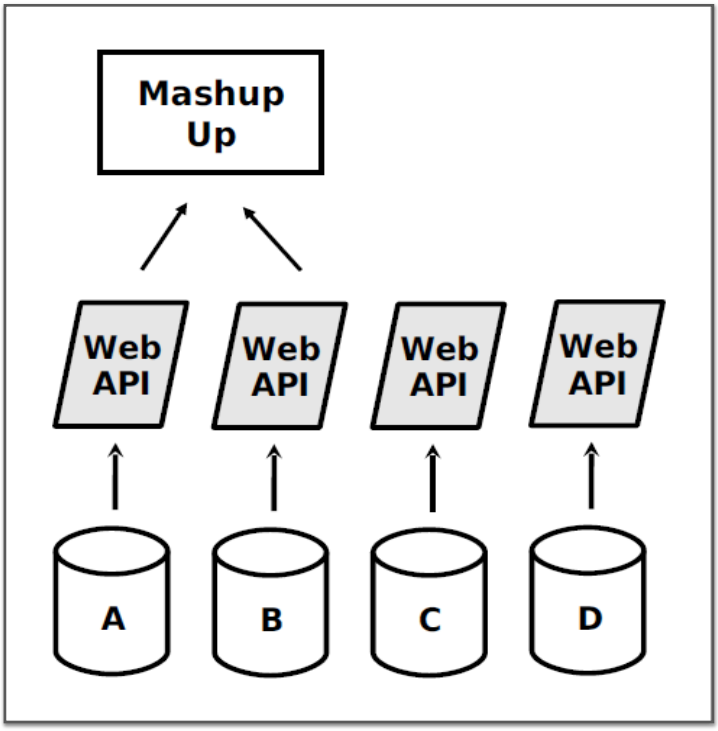


Figure 1.2 Web API's & Mashups
(From http://events.linkeddata.org/iswc2008tutorial/how-to-publish-linked-data-iswc2008-slides.pdf)

```
<div class="vevent">
  <span class="summary">bdigital</span>
  <abbr class="dtstart" title="2008-05-20">May 20</abbr> -

  <abbr class="dtend" title="2007-05-22">22</abbr>
</div>
```

Figure 1.3 Example of a Microformat.

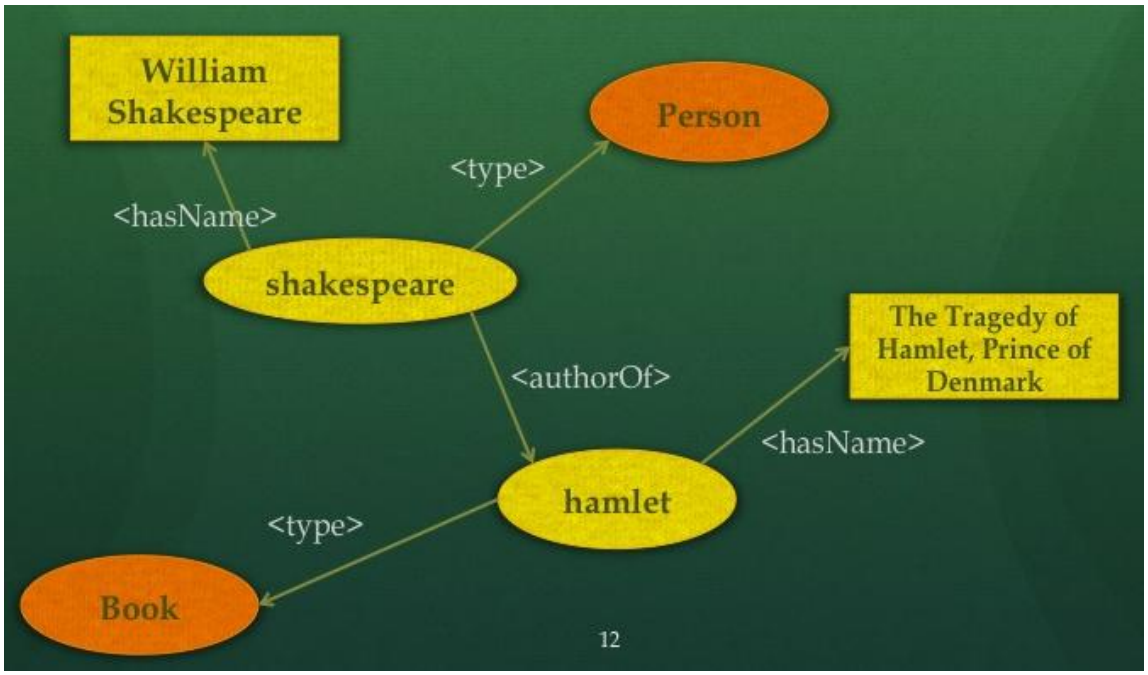


Figure 1.4 RDF graph describing Shakespeare

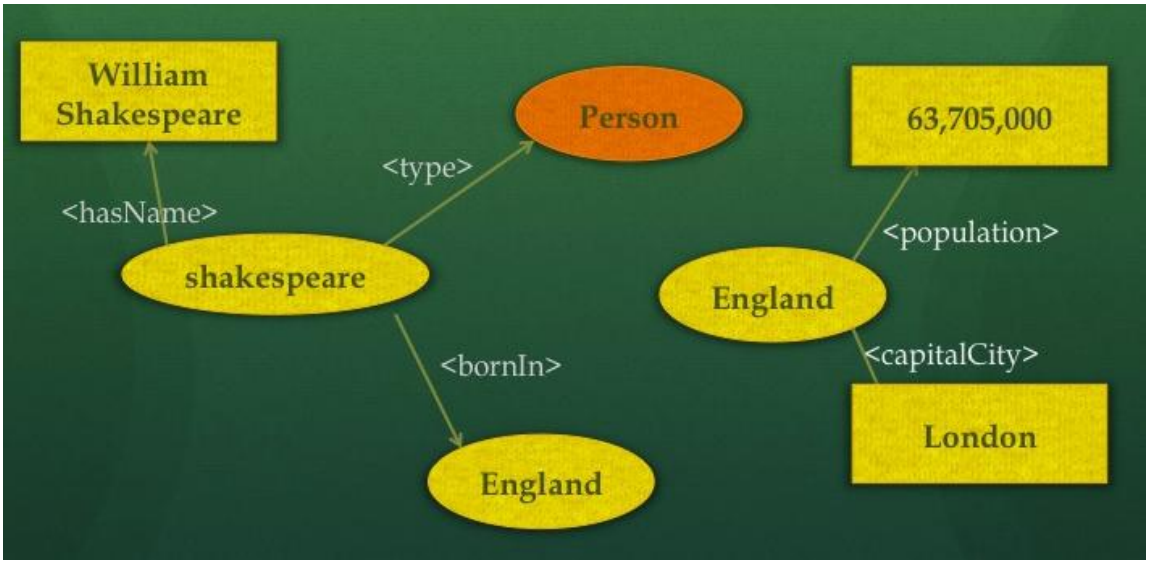


Figure 1.5 two RDF graphs before merging

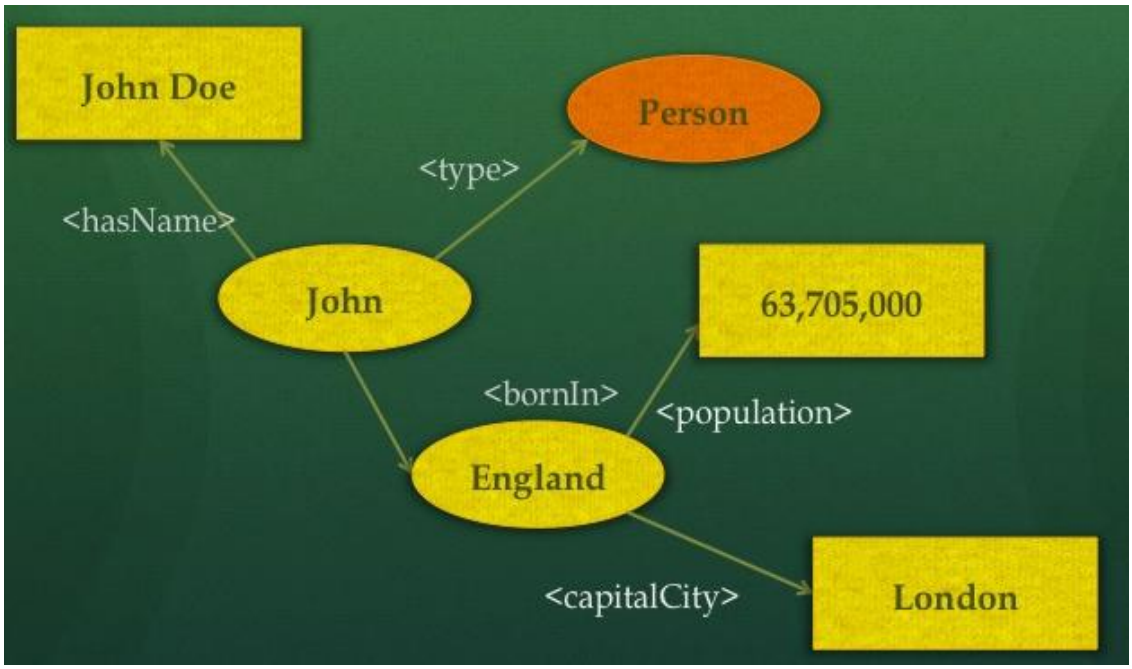


Figure 1.6 Merged RDF graph.

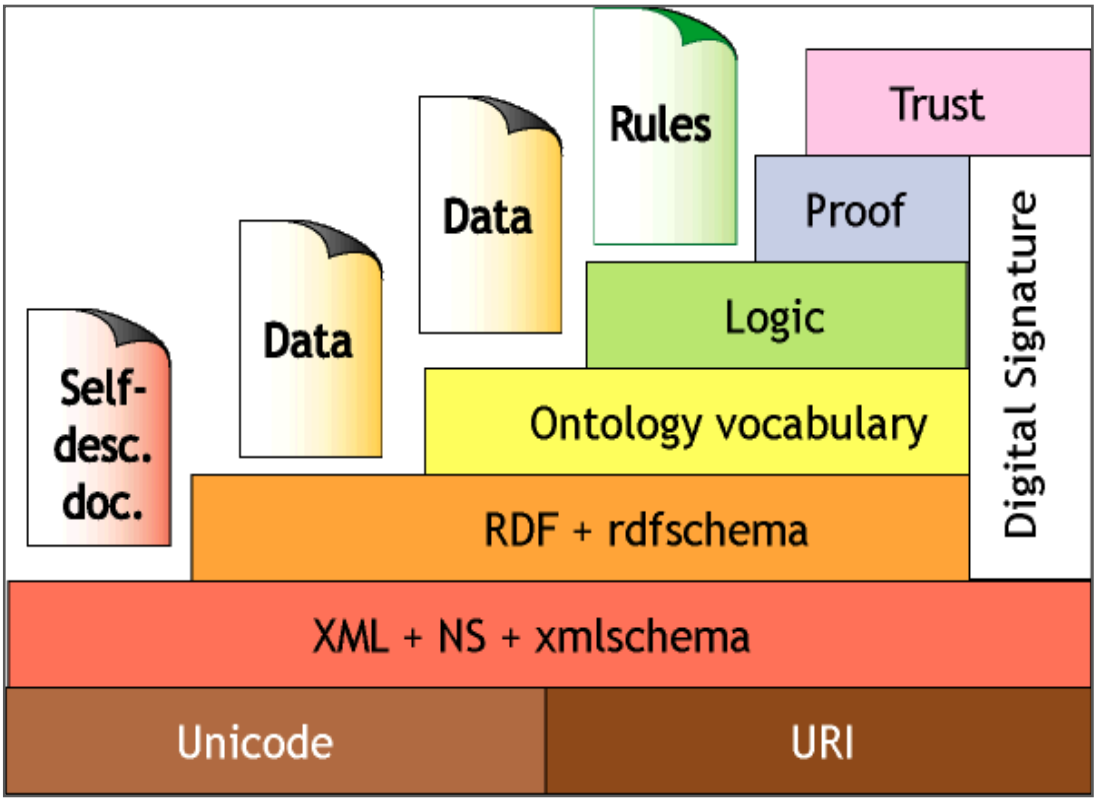


Figure: 1.7 Semantic Web Architecture

CHAPTER 2

PRIMARY AIM

The first step in publishing Linked Data as RDF is to select vocabularies to describe the data. One of the best practice is to choose existing vocabularies, because this enables existing applications to describe the data, this helps in making your data more interoperable with existing applications. But finding the existing vocabularies is difficult due to the following reasons.

1. No definitive place to search for existing vocabularies yet.
2. New vocabularies are being created and published every day, choosing one from is cumbersome for a human because he has to read the descriptions of all of them.
3. Often times, a single vocabulary does not cover all the entities and properties in our data, we have to mix and match multiple vocabularies.

The main functionality of the application is to provide vocabulary recommendations to users, who are willing to publish their data a linked data. The primary aim of this application is to help overcome these problems by automating the process of searching for existing vocabularies from multiple sources and also mix and match vocabularies.

CHAPTER 3

OVERALL DESIGN

The overall design of the application is shown in the figure 3.1. The application is made up of the following components.

1. User: who supplies all the entities and properties, that have to be modeled using RDF. The application mainly addresses two use cases. First use case is where the user manually provides all the information such as entities, properties and optionally the glosses describing the entities and properties.

   The other use case is where the user already has the data that he wants to model in a database so the user provides the database connection parameters and the application fetches the Table names i.e entities and column names i.e. properties, then the user can provide optional glosses.
2. Database: If we are trying to address the second use case above, data that has to be modeled is part of a database; metadata of database can be used to infer the entities and their properties.
3. Popular Vocabularies catalog, which is a catalog of existing vocabularies on the web along with their popularity.

This catalog keeps evolving currently this catalog only contains the list of vocabularies from the Linked Open Data Cloud. But the application is flexible enough to add new sources whenever needed.

4. Matcher: This will try to match the user/database input to the appropriate vocabulary terms. The matcher will return the results sorted based on the percentage of match from high to low. There are two types of matchers currently supported. The first type of matcher is a Distance Based matcher, that matches entities with classes and properties using distance based metrics like Levenshtein distance.

   The second type of matcher is a Semantic Matcher, which takes in to account the semantic similarity of strings. The Semantic Matcher matches based on 3 parameters. First the semantic similarity of entity/property from user input with class/property from vocabulary catalog. Second it considers the semantic similarity of glosses from input and catalog. Thirdly it considers the popularity of the vocabulary i.e. how many datasets in LOD cloud have been described using that vocabulary.

5. Testing: There are test cases for testing the correctness of both the backend and the UI. The UI test cases are written in Selenium, which is a framework for testing Angular JS. The backend is composed on web services and logic for matching which are written in Java. The correctness of these is tested using JUnit, Hamcrest and Mockito.

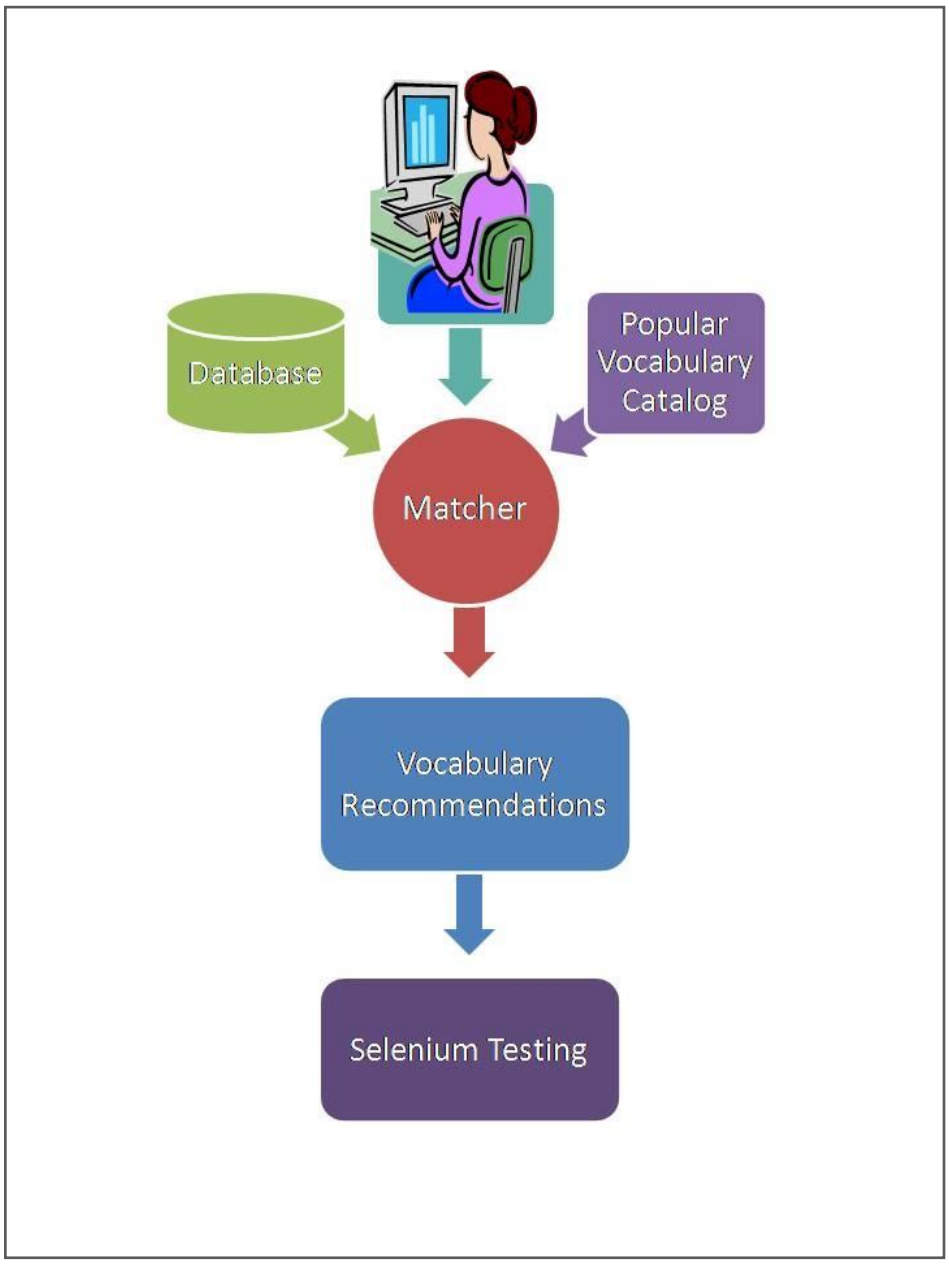


Figure 3.1 Overall Design

CHAPTER 4

DETAILED DESIGN

The figure 4.1 illustrates the detailed design of the vocabulary recommendation process. The entire process can be divided into 3 phases.

4.1 INPUT PREPROCESSING:

Careful analysis of current ontologies/vocabularies revealed that the names for majority of the entities are in either upper camel case or in underscore separated words. Similarly names of properties are in lower camel case or underscore separated words. So the input has to be transformed to these two formats.

Either a user can provide the input to the system or it can be extracted from a database. If a user provides the input, we assume that the data is already in processed and ready to use for matching. But if the input has to be extracted from a database, the input should be pre-processed before it is used for matching.

Input passes through a transformer, which can be configured to apply multiple transformation rules to transform the input values before comparison.

The transformer supports the following transformation rules

- removeSpecialChars - removes all special characters
- alphaReduce - removes everything except alphabets
- underscore - converts all words to lowercase.
- camelCase –converts all words to camel case.

4.2 BUILDING VOCABULARY CATALOG:

The process of building vocabulary is shown in the figure 4.2. Vocabulary Catalog is a collection of vocabularies and ontologies. The LOD stats website maintains comprehensive statistics about the datasets adhering to the RDF framework. The system will first connect to the LOD stats website and get a list of all vocabularies from it. Right now LOD has a list 543 vocabularies when we remove all the duplicates we have about 430 vocabularies.

For each vocabulary in this list, we run "schemagen" which is a tool provided apache Jena project. This tool converts the vocabulary or ontology in to a Java class file with a list of resources (classes) and properties.

This process takes a lot of time, to make this faster the system uses a thread pool that makes use of the multi core nature of the current processors and runs multiple instances of schemagen in multiple threads on each core and speeds up by an order of number of processors available on the machine on which the tool is running.

The system will then submit these java files to a vocabulary summarizer, which summarizes the vocabulary and allows us to get all the classes and properties defined in each vocabulary.

4.3 COMPARE & MATCH:

In this phase the input, which is a collection of entities and properties, will be matched with the collection of classes and properties from the vocabulary catalog that has been built and the results then displayed to the user in the order of similarity. The application provides two types of matchers a detailed discussion about them is provided below.

Distance Based Matcher:

The input to the matcher is a list of search keywords with an optional description for each keyword and a list of Resource/Property from the catalog. This matcher uses Levenshtein distance to compare search keyword with the object from catalog and if the similarity is greater than the threshold, then the catalog object is added to the results. Finally the results are ordered by the popularity of the vocabulary. Figure 4.3 gives an overview of how Distance Based Matcher works.

If the user provides a gloss/description for the entity or the property, then the gloss is compared with the description of the resource or the property. The results are then sorted based on the percentage of similarity. Any ties are broken based on the popularity of the vocabulary. If the user does not provide any

glosses, then the results are sorted by the match percentage and then the popularity of the vocabulary.

Semantic Matcher:

The input to the matcher is a list of search keywords with an optional description for each keyword and a list of Resource/Property from the catalog. This Matcher uses WordNet to semantically compare the search keyword and the resource/property. If the similarity is greater than the threshold, the resource/property is added to the list of results. The result includes the matched resource as well as the percentage of similarity. Figure 4.4 gives an overview of how Semantic Matcher works.

If the user provides a gloss for the entity or the property, then the gloss is compared with the description of the resource or the property. The results are then sorted based on the percentage of similarity. Any ties are broken based on the popularity of the vocabulary. If the user does not provide any glosses, then the results are sorted by the match percentage and then the popularity of the vocabulary. The results sorting process is shown in the figure 4.5.

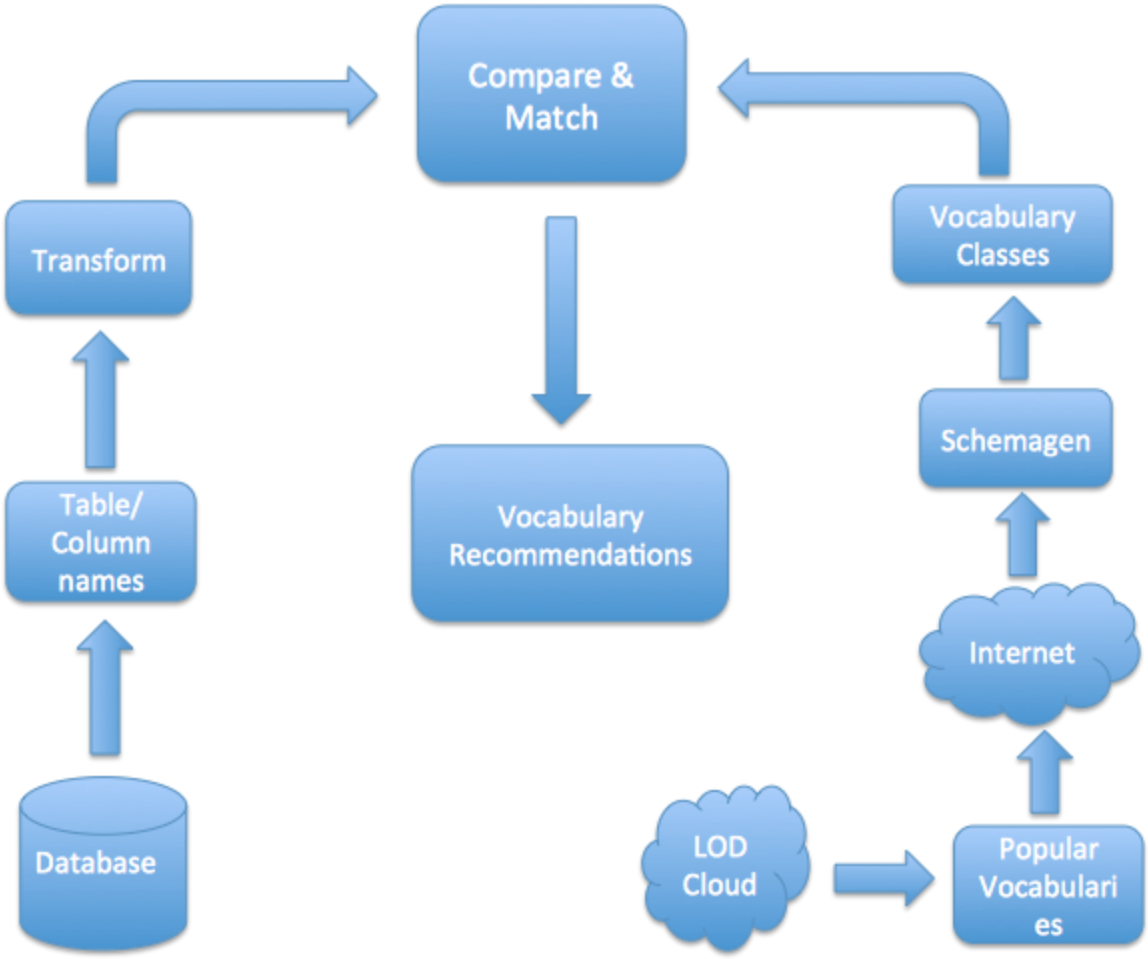


Figure 4.1: Detailed Design

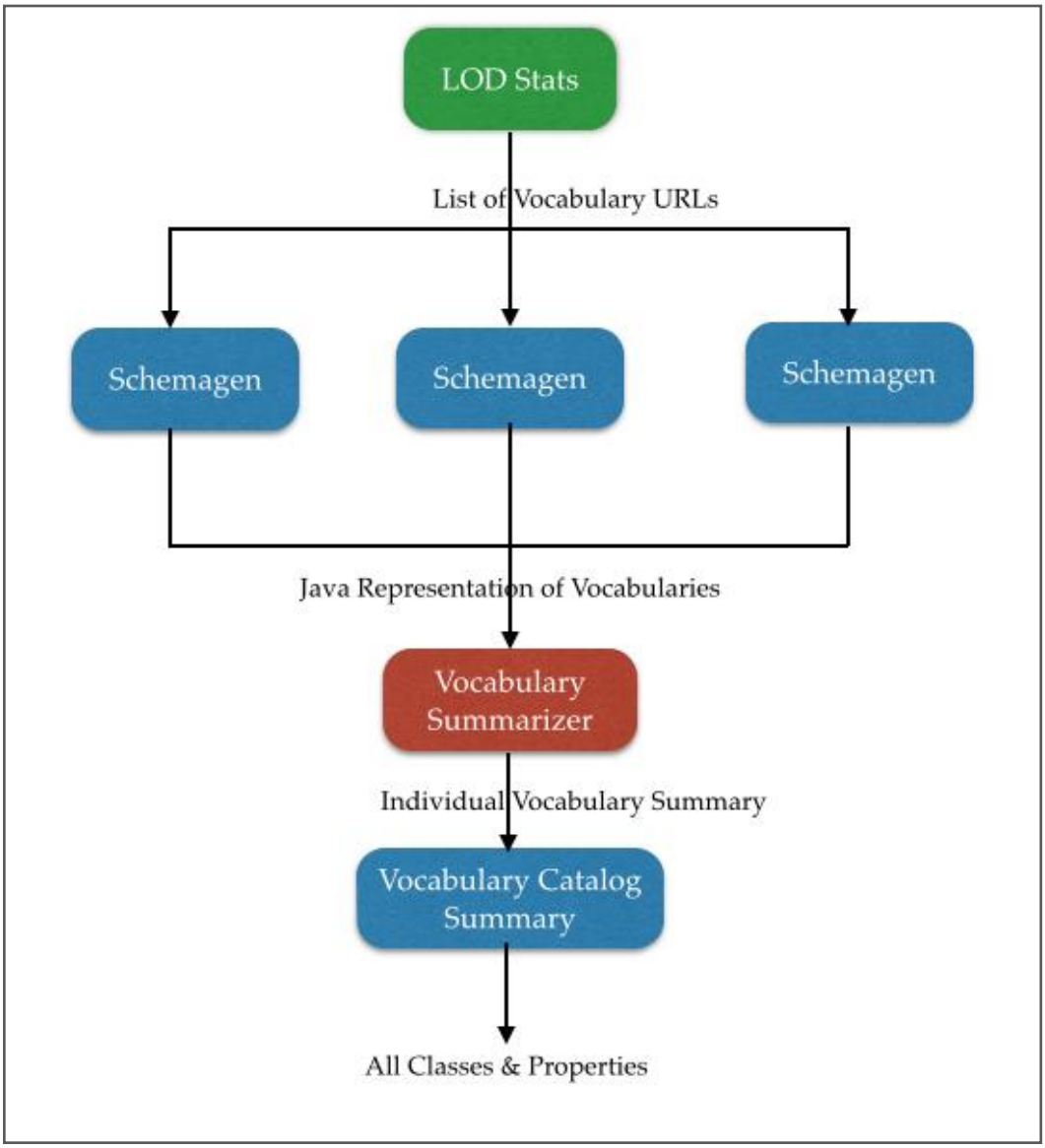


Figure 4.2: Process of building Vocabulary Catalog

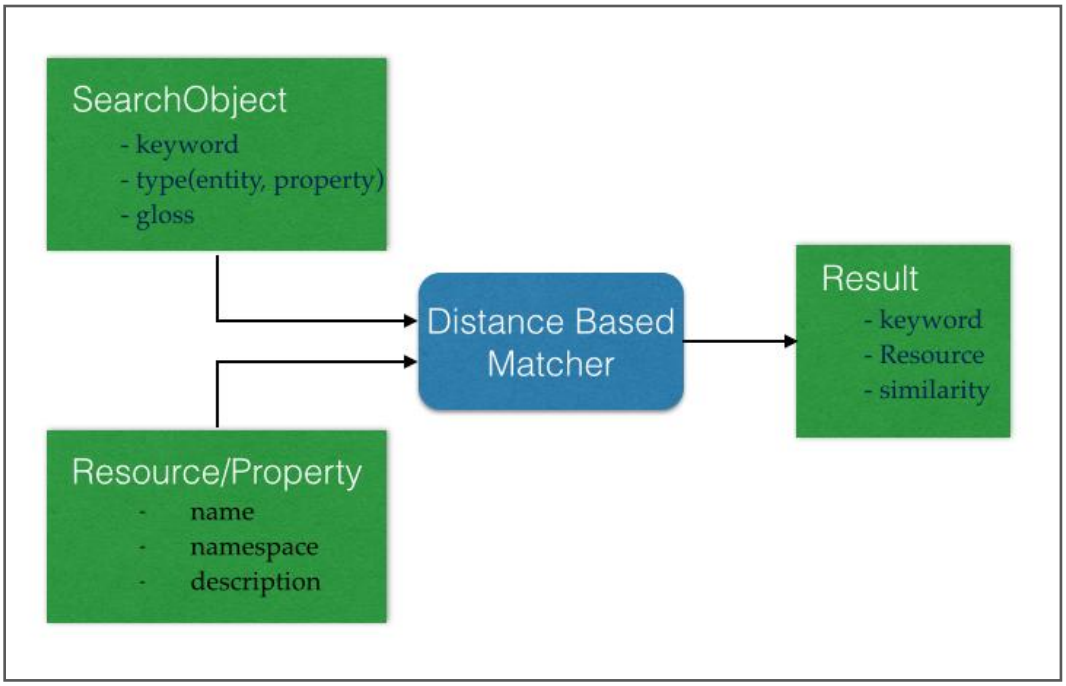


Figure 4.3: Distance Based Matcher

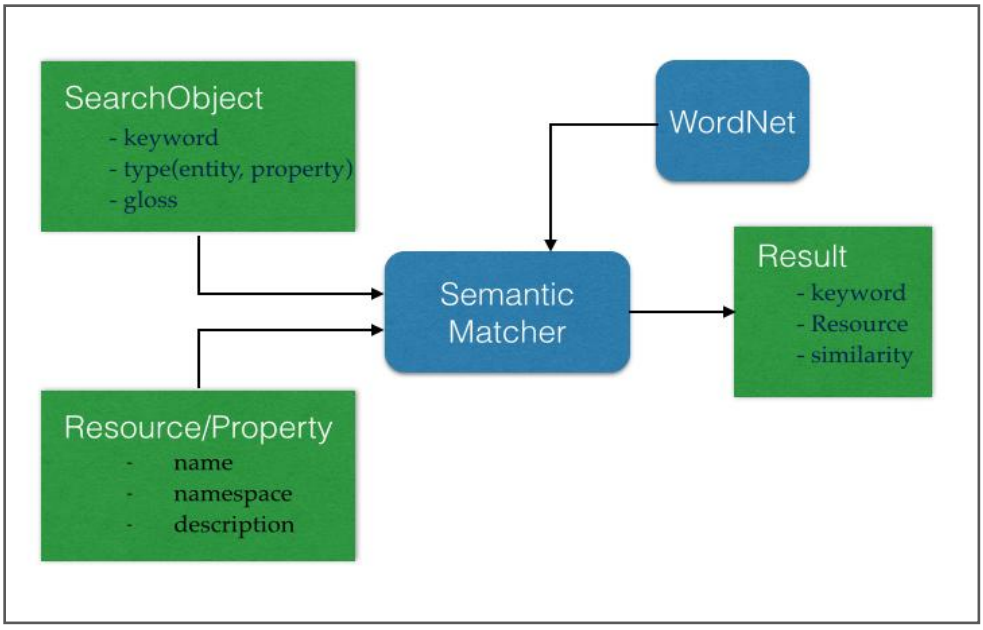


Figure 4.4: Semantic Matcher

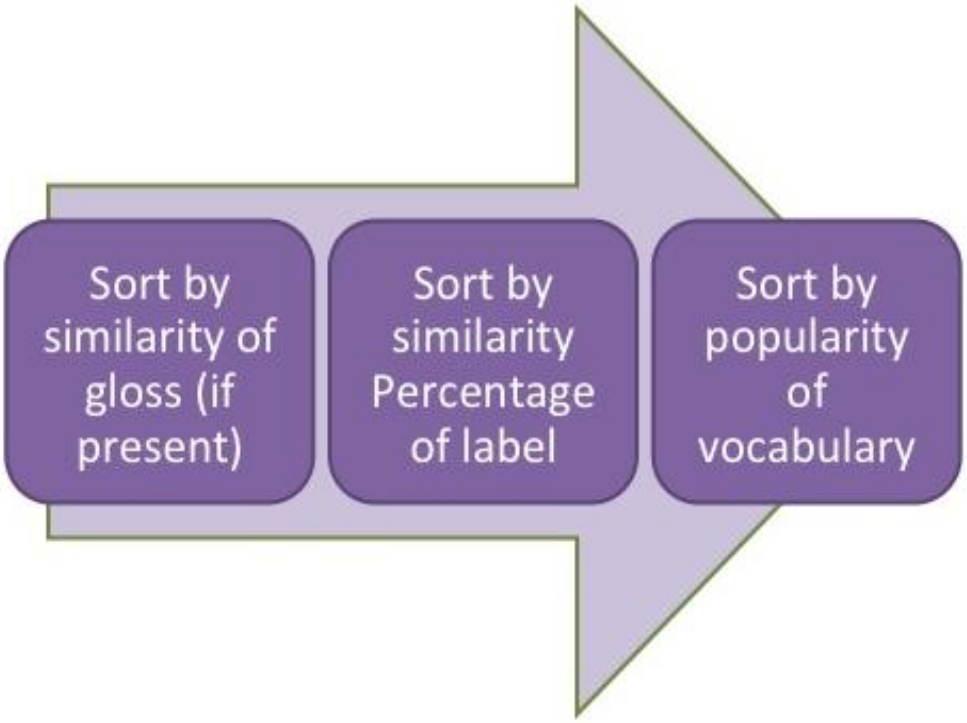


Figure 4.5 Process of sorting the match results

CHAPTER 5

TESTING

Two types of testing are performed to test the correctness of the application.

1. Functional Testing: In this type of testing, we will be testing the UI functionality of the application. This is performed using Selenium and JUnit. Functional testing involves the following steps. First we identify the functionality of the application. Then we create the inputs for the application. Execute the test case and compare the actual results to the expected results.

   Vocab4me has the following functionalities with sub functions that module performs.

   a. Generate Vocabulary Recommendations From Scratch

      i. Add Entity

      ii. Add Property

      iii. Get Recommendations by Semantic Matching

      iv. Get Recommendation by Distance Based Matching

b. Generate Vocabulary Recommendation from Database

i. Get Database Credentials and Display Table and Column Names

ii. Transform table & column names to required format

iii. Get Recommendations By Semantic Matching

iv. Get Recommendations By Distance Based Matching

All the functional test cases will be written using Selenium Web Driver for Java and JUnit. The following test cases are written to verify the correctness.

Add Entity: When this operation is performed, a new Entity has to be added to the tree structure. A test case is written to make sure that the Entity is being added as expected.

Add Property: When this operation is performed, a new Property has to be added to the selected node in the tree structure. A test case is written to make sure that the Property is being added as expected.

Get Recommendations By Distance Based Matching: When this operation is performed, the application should send the input entities and properties populated either manually by the user or by the populating them from the

database to the backend and the backend generates recommendations and sends it to the frontend and the results are displayed on the UI.

Get Recommendations By Semantic Matching: When this operation is performed, the application should send the input entities and properties populated either manually by the user or by the populating them from the database to the backend and the backend generates recommendations and sends it to the frontend and the results are displayed on the UI.

Get Database Credentials and Display Table and Column Names: The application takes the database credentials from the user and displays the table and column names on the UI. A test case is written to ensure this functionality.

Transform Table & Column Names: The application supports transforming the table and column names extracted from database to a format that is most suitable for matching entities. Application supports several different transformations. Test cases are written to make sure this functionality is as expected.

2. Unit Testing: In Unit testing, we will be performing testing of individual units such as classes are tested for their correctness. We will be performing unit testing of critical components such as DistanceBasedMatcher and SemanticMatcher.

CHAPTER 6

ENVIRONMENT

Software Requirements:

1. Operating System: Windows/Linux/Mac OS
2. Server: any Java web server
3. Programming Language: Java
4. Schemagen from Apache Jena Library
5. Web browser: any modern browser
6. Selenium Web Driver
7. Maven for build and dependency management.

CHAPTER 7

DEMONSTRATION

This section demonstrates how to use the application. The figure 7.1 shows the home page of the application. From this screen, the user can either choose to generate vocabulary recommendations from scratch or generate them from database

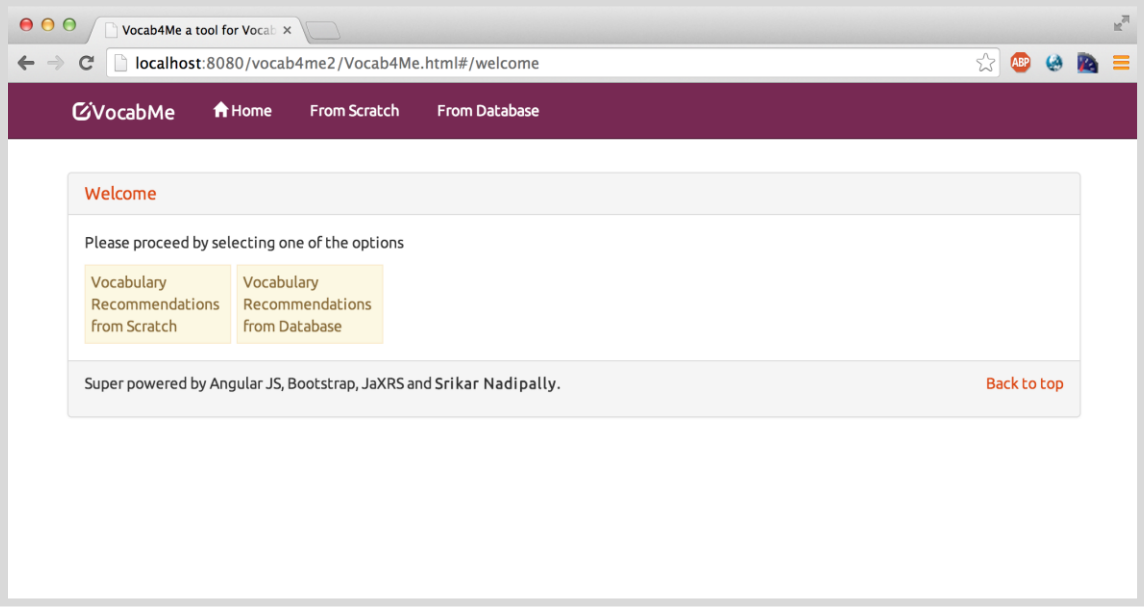
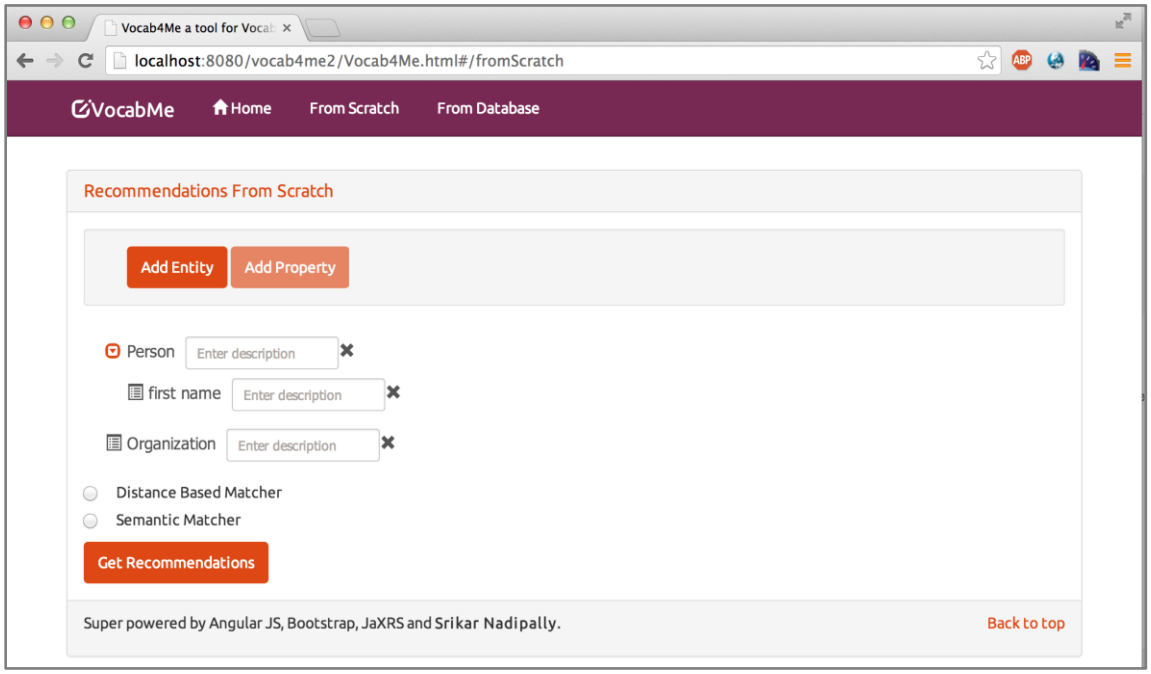


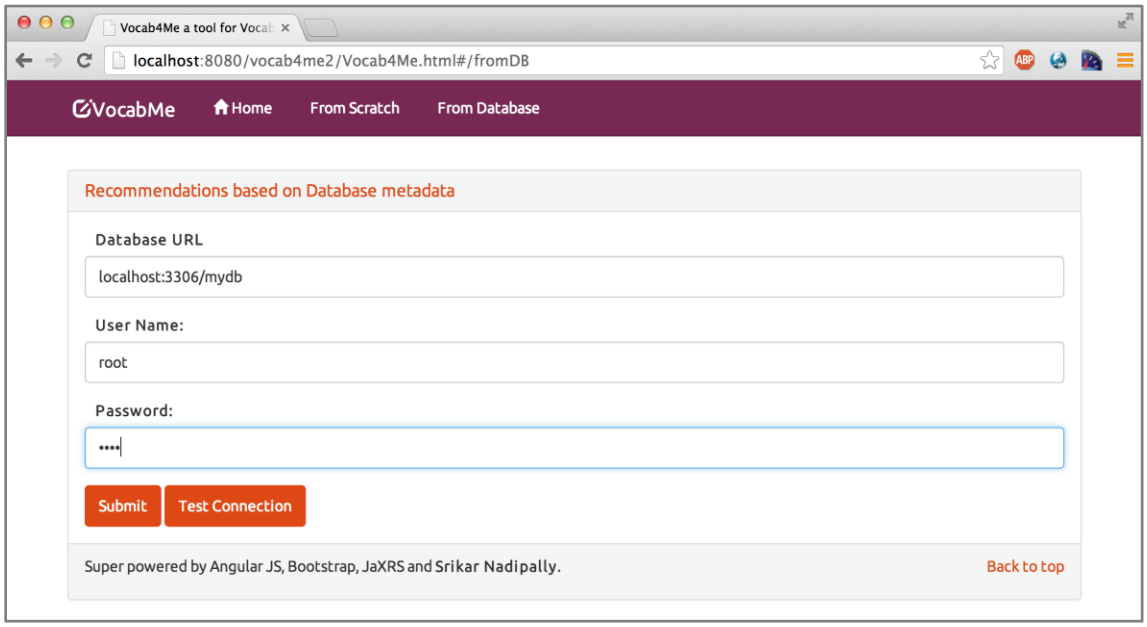
Figure 7.1 Initial Welcome screen

When the user selects generate recommendations from scratch, he will go to the page as shown in the figure 7.2. From here the user can add entities and properties manually, then he can choose to get recommendations using semantic matching or using Distance Based Matching.



Figures 7.2 UI Screen for generating recommendations from scratch

From the home screen when the user selects generate recommendations from database, he will go to the page as shown in the figure 7.3. Here the user can enter the database credentials and get the database table and column names. When the user presses submit and the credentials are correct, a screen very similar to Figure 7.4 will be shown. From here the user can transform the table and column names from Camel case to underscore notation and vice-versa and get recommendations.



Figures 7.3 UI Screen for entering database credentials

The figure 7.4 illustrates how the user can add new entities, when he chooses to generate recommendations from scratch. When the user adds a new entity, it will be shown in the tree structure shown on the page.

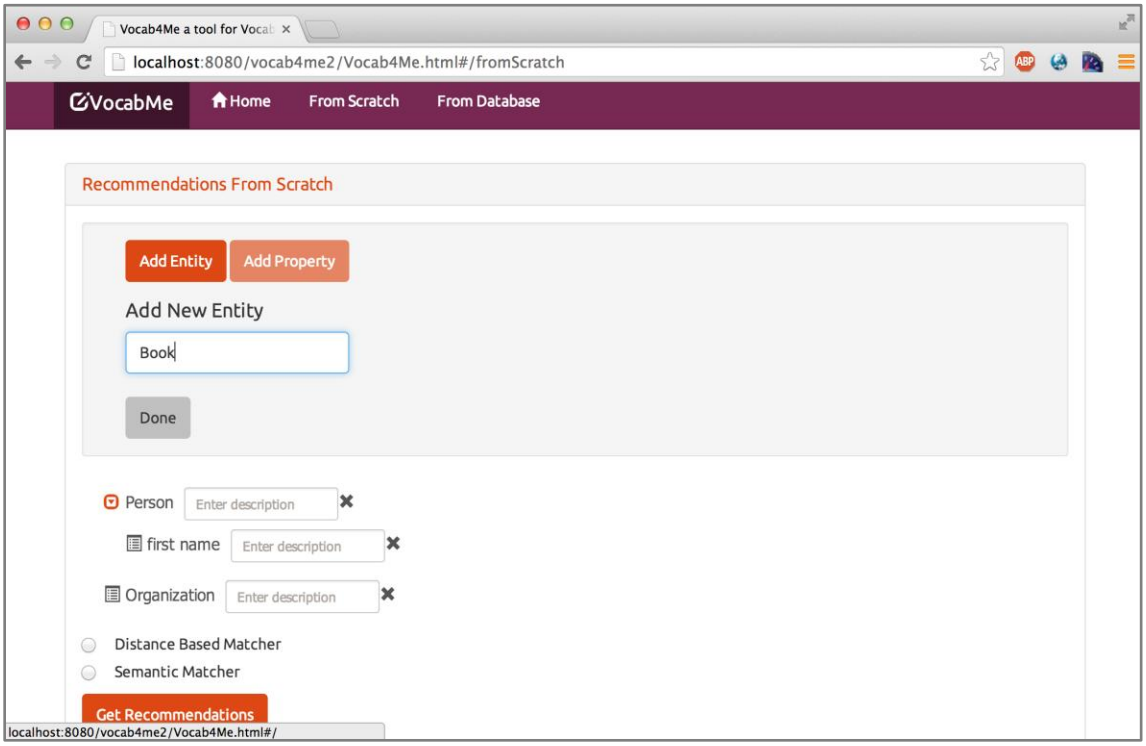


Figure 7.4 UI Screen to add new entities

The figure 7.5 illustrates how the user can add new properties, when he chooses to generate recommendations from scratch. The user has to select the entity node for which he is adding the property and add the property. When the user adds a new Property, it will be shown in the tree structure below the corresponding entity node the user selected.

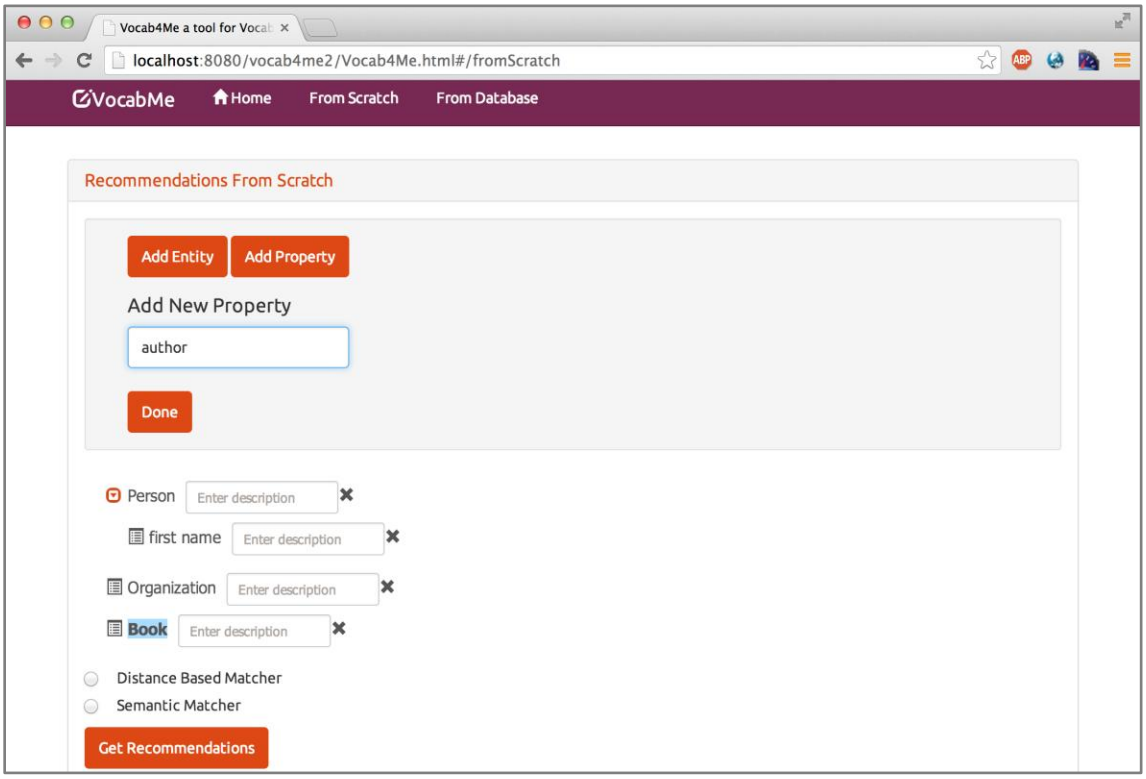


Figure 7.5 UI Screen to add new properties

The figure 7.6 shows the list of recommendations generated by the application for the Organization entity when the user chose Distance based matching.

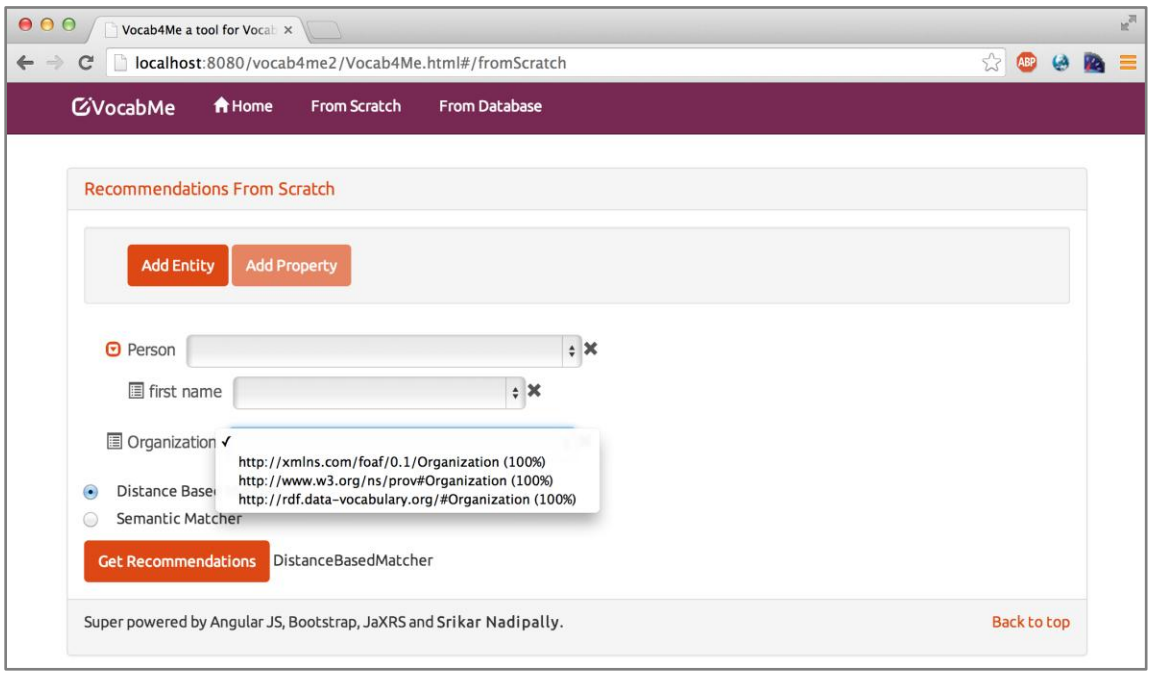


Figure 7.6 UI Screen displaying the list of suggestions for entity organization, when matching is done using Distance Based Matcher

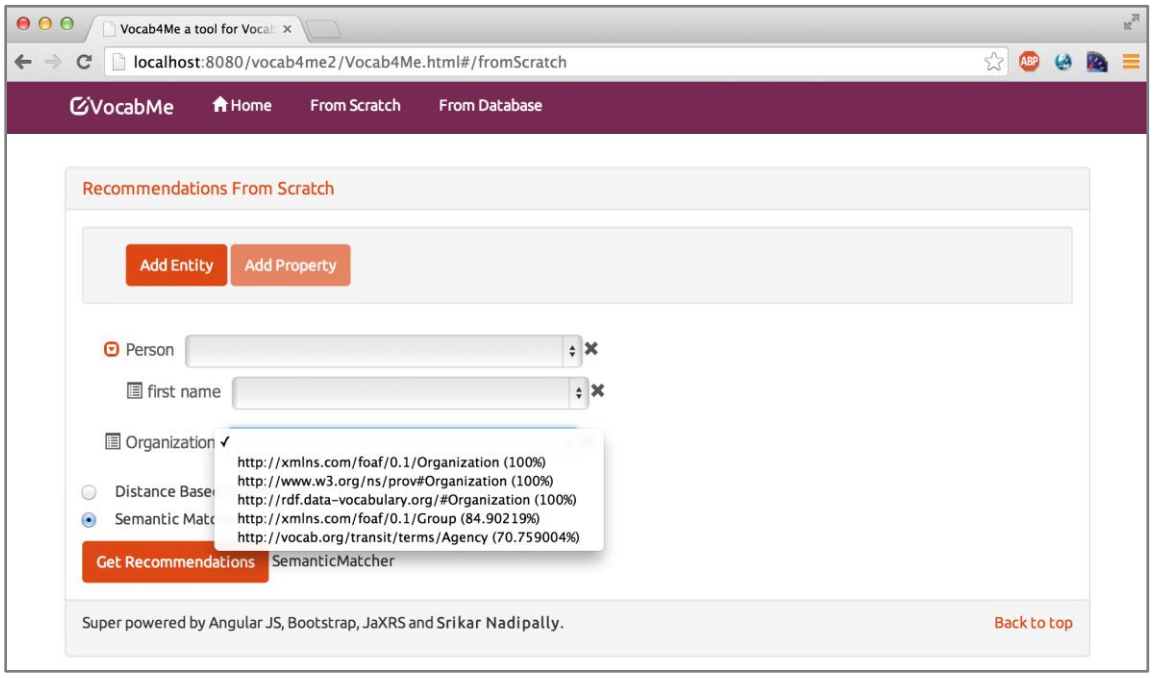


Figure 7.7 UI Screen displaying the list of suggestions for entity Organization, when matching is done using Semantic Matcher

REFERENCES

1. Allemang, Dean, and Jim Hendler. *SEMANTIC WEB for the WORKING ONTOLOGIST*. Morgan Kaufmann, 2008.
2. Heath, Tom, Christian Bizer, and James Hendler. *Linked Data Evolving the Web into a Global Data Space*. Morgan & Claypool Publishers, 2011.
3. Tom, Heath. Linked Data, "Linked Data: Frequently Asked Questions." Accessed November 19, 2013. http://linkeddata.org/faq.
4. Scientific American: Feature Article: The Semantic Web: May 2001.[ONLINE] Available at: http://www-sop.inria.fr/acacia/cours/essi2006/Scientific%20American_%20Feature%20Article_%20The%20Semantic%20Web_%20May%202001.pdf. [Accessed 27 February 2014].
5. Bernadette, Hyland, Ghislain Atemezing, Boris Villazón-Terrazas. W3C Working Group, "Best Practices for Publishing Linked Data." Accessed January 19, 2014. http://www.w3.org/TR/ld-bp/.
6. LODStats - 543 Vocabularies. 2014. LODStats - 543 Vocabularies. [ONLINE] Available at: http://stats.lod2.eu/vocabularies. [Accessed 24 February 2014].

7. Publishing information in the Linked Open Data cloud | The Bioinformatics Knowledgeblog. 2014. Publishing information in the Linked Open Data cloud | The Bioinformatics Knowledgeblog. [ONLINE] Available at: http://bioinformatics.knowledgeblog.org/2011/07/05/publishing-information-in-the-linked-open-data-cloud/. [Accessed 24 February 2014].
8. Resource Description Framework (RDF): Concepts and Abstract Syntax. 2014. Resource Description Framework (RDF): Concepts and Abstract Syntax. [ONLINE] Available at: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/. [Accessed 24 February 2014].
9. Semantic Web - XML2000 - Slide list. 2014. Semantic Web - XML2000 - Slide list. [ONLINE] Available at: http://www.w3.org/2000/Talks/1206-xml2k-tbl/. [Accessed 24 February 2014].
10. Inference - W3C. 2014. Inference - W3C. [ONLINE] Available at: http://www.w3.org/standards/semanticweb/inference.html. [Accessed 24 February 2014].
11. Apache Jena - Jena schemagen HOWTO. 2014. [ONLINE] Available at: http://jena.apache.org/documentation/tools/schemagen.html. [Accessed 24 February 2014].
12. The D2RQ Platform – Accessing Relational Databases as Virtual RDF Graphs. 2014. [ONLINE] Available at: http://d2rq.org/. [Accessed 24 February 2014].

13. Silk Link Discovery Framework Project | Assembla. 2014. [ONLINE] Available at: https://www.assembla.com/spaces/silk/wiki. [Accessed 24 February 2014].

APPENDIX A

FUTURE SCOPE

1. Integrate the application with D2RQ:

   Once the user finalizes all the recommendations provided by the applications, we can generate mapping file that uses these recommendations and fully automate publishing linked data from relational databases.

2. Save recommendations to a file and open existing recommendations

APPENDIX B

Installation Instructions

Download the source of vocab4me from the following URL https://drive.google.com/file/d/0BwxhXpW7hFEyVFlWM015YUdzeUE/edit?usp=sharing

Extract the zip file of vocab4me2 and in the command prompt or terminal go to the directory containing the pom.xml

Install Maven and add maven to the operating system path.

Then run the following two commands to install library files that are not part of maven central repository.

1. mvn install:install-file -Dfile=src/main/webapp/WEB-INF/lib/ws4j-1.0.1.jar -DgroupId=com.ws4j -DartifactId=ws4j -Dversion=1.0.1 -Dpackaging=jar -DgeneratePom=true
2. mvn install:install-file -Dfile=src/main/webapp/WEB-INF/lib/simmetrics1_6_2.jar -DgroupId=com.simmetrics -DartifactId=simmetrics -Dversion=1.6.2 -Dpackaging=jar -DgeneratePom=true

After this step run the following command and check to see if the project is setup correctly.

```
mvn compile
```

If everything works out fine run the following command that will start a tomcat server, deploy the application to tomcat.

```
mvn tomcat:run
```

If everything went well if we navigate to http://localhost:8080/vocab4me2/Vocab4Me.html in your browser, you should be able to see the application up and running.

APPENDIX C

LIST OF POPULAR VOCABULARIES

1. http://purl.org/dc/elements/1.1/
2. http://www.w3.org/2004/02/skos/core#
3. http://www.w3.org/2003/01/geo/wgs84_pos#
4. http://www.w3.org/1999/xhtml/vocab#
5. http://www.aktors.org/ontology/portal#
6. http://purl.org/ontology/bibo/
7. http://purl.org/ontology/mo/
8. http://www.w3.org/2006/vcard/ns#
9. http://rdfs.org/sioc/ns#
10. http://creativecommons.org/ns#
11. http://www.geonames.org/ontology#
12. http://purl.org/vocab/frbr/core#
13. http://www.w3.org/2001/XMLSchema#
14. http://www.w3.org/2006/time#
15. http://purl.org/NET/c4dm/event.owl#
16. http://dbpedia.org/resource/
17. http://purl.org/goodrelations/v1#

18. http://dbpedia.org/ontology/

19. http://purl.org/vocab/bio/0.1/

20. http://dbpedia.org/property/

21. http://www.holygoat.co.uk/owl/redwood/0.1/tags/

22. http://rdfs.org/ns/void#

23. http://purl.org/NET/scovo#

24. http://www.w3.org/2006/http#

25. http://purl.uniprot.org/core/

26. http://umbel.org/umbel#

27. http://purl.org/stuff/rev#

28. http://purl.org/linked-data/cube#

29. http://rdf.geospecies.org/ont/geospecies#

30. http://purl.org/linked-data/sdmx#

31. http://www.w3.org/ns/sawsdl#

32. http://www.w3.org/ns/org#

33. http://purl.org/vocab/vann/

34. http://data.ordnancesurvey.co.uk/ontology/admingeo/

35. http://www.w3.org/2007/05/powder-s#

36. http://usefulinc.com/ns/doap#

37. http://lod.taxonconcept.org/ontology/txn.owl#

38. http://xmlns.com/wot/0.1/

39. http://purl.org/net/compass#

40. http://www.w3.org/2004/03/trix/rdfg-1/

41. http://purl.org/NET/c4dm/timeline.owl#

42. http://purl.org/dc/dcam/

43. http://swrc.ontoware.org/ontology#

44. http://zeitkunst.org/bibtex/0.1/bibtex.owl#

APPENDIX D

LIBRARIES & FRAMEWORKS USED

1. Apache Jena framework for using Schemgen
2. Angular JS, which is a MVC JavaScript framework for UI.
3. Apache Commons
4. Google Guava
5. Jackson, which is an implementation of JAXRS restful web services in Java
6. GSON, JSON processing library from Google
7. Simmetrics - which contains implementation of Distance based matching algorithms
8. WS4J - which contains implementation of semantic matching algorithms
9. JUnit, Mockito & Hamcrest for unit testing.

APPENDIX E

SELENIUM TEST CASES

These test cases are written for chrome browser. They use the chrome driver for executing the test cases. The following is a listing of the support classes and selenium test classes.

MyChromeDriver.java

```
package com.vocab4me.selenium;

import org.openqa.selenium.chrome.ChromeDriver;

import com.vocab4me.util.Utils;

public class MyChromeDriver extends ChromeDriver{

	static{

		System.setProperty("webdriver.chrome.driver",

Utils.getResourcesDirectory().getAbsolutePath()+"/chromedriver");

	}


	public static final String HOME_URL =

"http://localhost:8080/vocab4me2/Vocab4Me.html";

}
```

HomePage.java

```
package com.vocab4me.selenium;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.FindBy;

public class HomePage {

	public HomePage(WebDriver driver) {

		this.driver = driver;

		driver.get(MyChromeDriver.HOME_URL+"#/welcome");

	}

	public String getCurrentURL(){

		return driver.getCurrentUrl();

	}

	public void navigateToFromScratchPage(){

		fromScratchPage.click();

	}

	public void navigateToFromDBPage(){

		fromDBPage.click();

	}


	public void close(){
```

```
            driver.close();
        }

        @FindBy(name="fromScratchPage") WebElement fromScratchPage;
        @FindBy(name="fromDBPage") WebElement fromDBPage;

        private final WebDriver driver;
}
```

HomePageTest.java

```
package com.vocab4me.selenium;

import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.*;
import static org.junit.Assert.*;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import org.openqa.selenium.support.PageFactory;
```

```
public class HomePageTest {
	private static HomePage homePage;

	@BeforeClass
	public static void openTheBrowser() {
		homePage = PageFactory.initElements(new MyChromeDriver(),
HomePage.class);
	}
	@Test
	public void testHomePageToFromScratchNavigation() throws Exception {
		homePage.navigateToFromScratchPage();
		Thread.sleep(2000);
		assertThat(homePage.getCurrentURL(),
equalTo(MyChromeDriver.HOME_URL+"#/fromScratch"));
	}
	@Test
	public void testHomePageToFromDBNavigation() throws Exception {
		homePage.navigateToFromDBPage();
		Thread.sleep(2000);
		assertThat(homePage.getCurrentURL(),
equalTo(MyChromeDriver.HOME_URL+"#/fromDB"));
```

```
	}
	@AfterClass
	public static void closeTheBrowser() {
		homePage.close();
	}
}
```

FromScratchPage.java

```
package com.vocab4me.selenium;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.*;
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.NoSuchElementException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.FindBys;
import org.openqa.selenium.support.ui.FluentWait;
import org.openqa.selenium.support.ui.Wait;
```

```
import com.google.common.base.Function;
import com.google.common.base.Predicate;
import com.google.common.collect.Collections2;
import com.google.common.collect.Lists;

public class FromScratchPage {

	@FindBy(id="addEntity") private WebElement addEntityButton;
	@FindBy(id="addProperty") private WebElement addPropertyButton;
	@FindBy(id="entityDone") private WebElement addEntityDoneButton;
	@FindBy(id="propertyDone") private WebElement
addPropertyDoneButton;
	@FindBy(id="matcher1") private WebElement distanceBasedMatcher;
	@FindBy(id="matcher2") private WebElement semanticMatcher;
	@FindBy(id="getRecommendations") private WebElement
getRecommendationsButton;
	@FindBy(id="tempEntityName") private WebElement
newEntityInputBox;
	@FindBy(id="tempPropertyName") private WebElement
newPropertyInputBox;

	@FindBys(@FindBy(css="div #treeView > li")) private List<WebElement>
```

```
entities;
        @FindBys(@FindBy(css="div[name='children'] #treeView > li")) private
List<WebElement> properties;
        private WebDriver driver;

        public FromScratchPage(WebDriver driver) {
                this.driver = driver;
                driver.get(MyChromeDriver.HOME_URL+"#/fromScratch");
        }

        public void addEntity(String name){
                assertThat(newEntityInputBox.isDisplayed(), equalTo(false));
                addEntityButton.click();
                assertThat(newEntityInputBox.isDisplayed(), equalTo(true));
                newEntityInputBox.sendKeys(name);
                addEntityDoneButton.click();
                assertThat(newEntityInputBox.isDisplayed(), equalTo(false));
        }

        public void addProperty(String entityName, String propertyName){
                assertThat(addPropertyButton.isEnabled(), equalTo(false));
                WebElement element = findEntityWithName(entityName);
```

```
            element.findElement(By.tagName("span")).click();
            assertThat(addPropertyButton.isEnabled(), equalTo(true));

            assertThat(newPropertyInputBox.isDisplayed(), equalTo(false));
            addPropertyButton.click();
            assertThat(newPropertyInputBox.isDisplayed(), equalTo(true));

            newPropertyInputBox.sendKeys(propertyName);
            addPropertyDoneButton.click();
            assertThat(newPropertyInputBox.isDisplayed(), equalTo(false));

        }

        private WebElement findEntityWithName(String entityName) {
            for(WebElement entity : entities){

        if(entity.findElement(By.tagName("span")).getText().equals(entityName)){
                        return entity;
                    }
            }
            //throw new NoSuchElementException("Entity with the specified
name does not exist");
```

```
            return null;
        }

        public void selectMatcher(String name){
            if(name.equalsIgnoreCase("semanticmatcher")){
                semanticMatcher.click();
            }else{
                distanceBasedMatcher.click();
            }
        }

        public void clickOnGetRecommendations() {
            if(matcherNotSelected()){
                throw new RuntimeException("Matcher should be selected
for recommendations");
            }
            getRecommendationsButton.click();

            WebElement element =
driver.findElement(By.cssSelector("#treeView li select"));

            while(!element.isDisplayed()){
```

```
                try {
                        Thread.sleep(1000);
                } catch (InterruptedException e) {}
        }

    }

    private boolean matcherNotSelected() {
        return !(semanticMatcher.isSelected() ||
distanceBasedMatcher.isSelected());
    }

    public void close() {
        driver.close();

    }

    public List<String> getRecommendationsFor(String entityName) {

        WebElement entityElement = findEntityWithName(entityName);
```

```
            List<WebElement> recommendationElements =
entityElement.findElements(By.cssSelector("select")).get(0).findElements(By.tagN
ame("option"));

            List<String> recommendations = Lists.newArrayList();

            for(WebElement recommendation : recommendationElements){
                  recommendations.add(recommendation.getText());
            }
            return recommendations;

      }


      public List<String> getPropertiesForEntity(String entityName){
            List<String> properties = Lists.newArrayList();

            WebElement entityElement = findEntityWithName(entityName);
            List<WebElement> propElements =
entityElement.findElements(By.cssSelector("li"));
```

```
            for(WebElement prop : propElements){
                properties.add(prop.getText());
            }

            return properties;
        }

        public List<String> getEntities(){
            List<String> entitiesStr = Lists.newArrayList();

            for(WebElement entity : entities){
                entitiesStr.add(entity.getText());
            }

            return entitiesStr;
        }

        public void refresh() {
            driver.navigate().refresh();

        }
}
```

FromScratchPageTest.java

```
package com.vocab4me.selenium;

import static org.hamcrest.MatcherAssert.*;
import static org.hamcrest.Matchers.*;
import static org.junit.Assert.assertTrue;

import java.util.List;

import org.apache.commons.lang3.StringUtils;
import org.apache.commons.lang3.SystemUtils;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import org.openqa.selenium.support.PageFactory;

import com.hp.hpl.jena.reasoner.rulesys.builtins.GreaterThan;

public class FromScratchPageTest {
```

```
	private static FromScratchPage fromScratchPage;

	@BeforeClass
	public static void openTheBrowser() {
		fromScratchPage = PageFactory.initElements(new
MyChromeDriver(), FromScratchPage.class);

	}

	@Before
	public void refresh(){
		fromScratchPage.refresh();
	}

	@Test
	public void testAddEntity() throws Exception {
		fromScratchPage.addEntity("Book");
		List<String> entities = fromScratchPage.getEntities();
		assertThat(entities, hasSize(greaterThan(0)));
		assertThat(entities.get(entities.size() - 1), equalTo("Book"));
	}
```

```
	@Test
	public void testAddProperty() throws Exception {
		fromScratchPage.addEntity("Book");
		fromScratchPage.addProperty("Book", "authorName");
		List<String> props =
fromScratchPage.getPropertiesForEntity("Book");

		assertThat(props, hasSize(greaterThan(0)));
		assertThat(props.get(props.size() - 1), equalTo("authorName"));
	}

	@Test
	public void testGetRecommendationsSemanticMatcher() throws
Exception {
		fromScratchPage.selectMatcher("semanticMatcher");
		fromScratchPage.clickOnGetRecommendations();
		Thread.sleep(1000);
		List<String> recommendationsFor =
fromScratchPage.getRecommendationsFor("Person");

		assertThat(recommendationsFor.size(), greaterThan(0));
```

```
	}

	@Test
	public void testGetRecommendationsDistanceBasedMatcher() throws
Exception {
		fromScratchPage.selectMatcher("distanceBasedMatcher");
		fromScratchPage.clickOnGetRecommendations();
		Thread.sleep(1000);
		List<String> recommendationsFor =
fromScratchPage.getRecommendationsFor("Person");

		assertThat(recommendationsFor.size(), greaterThan(0));

		for(String recommendation : recommendationsFor){
			if(recommendation.isEmpty()){
				continue;
			}
			assertThat(StringUtils.containsIgnoreCase(recommendation,
"person"), equalTo(true));
		}

	}
```

```
	@AfterClass
	public static void closeTheBrowser() {
		fromScratchPage.close();
	}
}
```