PROYECTO FIN DE CARRERA

Título:	Design and implementation of an HTML5 Framework for biodiversity and environmental information visualization based on Geo Linked Data
Autor:	Rubén Díaz Vega
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	Mercedes Garijo Ayestarán
Vocal:	Tomás Robles Valladares
Secretario:	Carlos Ángel Iglesias Fernández
Suplente:	Marifeli Sedano Ruíz

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

PROYECTO FIN DE CARRERA

DESIGN AND IMPLEMENTATION OF AN HTML5 FRAMEWORK FOR BIODIVERSITY AND ENVIRONMENTAL INFORMATION VISUALIZATION BASED ON GEO LINKED DATA

Rubén Díaz Vega

Diciembre de 2014

Resumen

Este documento contiene los resultados de un proyecto cuyo principal objetivo es desarrollar un Framework HTML5 que permita la consulta, el tratamiento y la representación gráfica de conjuntos de datos de la Web Semántica desde un punto de vista geográfico.

En primer lugar, analizamos el estado actual de los *Datos Enlazados Geográficos* (*Geo Linked Data*) y el gran crecimiento que ha experimentado en los últimos años. Tras esto, presentamos la necesidad de desarrollar una aplicación web que nos permita consultar estos conjuntos de datos disponibles en la web y tratar, indexar, filtrar y representar la información consultada. Esta aplicación nos ayudaría a sacar mayor provecho de todas las ventajas que nos ofrece la información geográfica disponible en la Web Semántica.

Una vez definido nuestro objetivo, analizamos las diferentes herramientas que utilizaremos para desarrollar nuestra aplicación. Hemos elegido una aplicación web desarrollada por el *Grupo de Sistemas Inteligentes* (GSI[1]) como punto de partida. Sobre esta aplicación introduciremos las características y herramientas necesarias para trabajar con Sistemas de Información Geográfica (GIS).

En capítulos posteriores, presentaremos la arquitectura del sistema. Hemos optado por una arquitectura modular de forma que cada uno de los módulos que componen la aplicación tenga unas funciones bien definidas, de manera que el mantenimiento y el desarrollo futuro sobre la aplicación sean más fáciles. Además, en otro capítulo incluirémos la experimentación con la aplicacion, detallando los diferentes tests llevados a cabo, sus resultados y el impacto que han tenido en nuestras decisiones.

Finalmente, presentamos las conclusiones del trabajo realizado y el posible trabajo futuro que podría llevarse a cabo para mejorar el proyecto.

Palabras clave: Tecnologías Semánticas, Linked data, Sefarad, RDF, SPARQL, PHP, JavaScript, Java, Knockout JS, Geográfico

[1]http://www.gsi.dit.upm.es/

Abstract

This work collects the results of a project whose main purpose is to develop an HTML5 framework to query Linked Data datasets and manage and display the retrieved data from a geographical perspective.

First, we analyse the current state of *Geo Linked Data* and the great growth experienced in recent years. After that, we present the need to develop a web application that allows us to query any dataset available on the web and manage, index, filter and display the retrieved data. This application could help us to take more advantage of the benefits of the geographical information available on the Semantic Web.

After defining our goal, we analyse the different tools available which will help us to develop our application. We chose to improve a web application develop by *Group of Intelligent Systems* (GSI[2]) and introduce the needed features to work with GIS datasets.

Furthermore, we present the architecture of the system, which is based on different modules in order to have a modular structure that facilitates the development and the contribution of future developers. We have a specific chapter for the experimentation where it is presented the different tests executed as well as the results of them and its impact in our decisions.

Finally, we present the conclusions of the work and the possible future work that could be done in order to improve the project.

Keywords: Semantic technologies, Linked data, Sefarad, RDF, SPARQL, PHP, JavaScript, Java, Knockout JS, Geo

[2]http://www.gsi.dit.upm.es/

Agradecimientos

En primer lugar, a mis padres. Gracias a las posibilidades y oportunidades brindadas he podido estudiar la carrera que quería y, en consecuencia, desarrollar este proyecto. Gracias a mi madre por el apoyo dado y por preocuparse por mí por encima de todas las demás cosas en los momentos difíciles. Gracias mamá.

Gracias a mis abuelos por interesarse por mí día tras día. Especialmente a mi abuela, por preguntar por mí y rezar porque todo vaya bien y sigamos avanzando noche tras noche. Espero que quién sea que escuche tus oraciones aguante un poco más, que ahora viene lo difícil: la vida laboral. Gracias abuela.

Gracias a mi vida, Mónica. Gracias por estar siempre a mi lado. Gracias por compartir cada momento conmigo. Gracias por celebrar mis logros conmigo y apoyarme, animarme e inspirarme en los momentos malos. Al igual que en todo lo que rodea mi día a día, has sido una parte fundamental en mi éxito en esta aventura. Gracias cariño.

Gracias a otro de los pilares de mi vida: los buenos amigos. Gracias a los amigos de siempre, los que me acompañan desde que era un enano y han seguido ahí año tras año, y continúan hasta hoy. Gracias también a los nuevos amigos que he conocido en estos años de universidad. Ha sido una experiencia única y, en gran medida, ha sido gracias a vosotros.

Dar las gracias también a los compañeros del departamento. Este año trabajando en el grupo no hubiese sido lo mismo sin vosotros. Ha sido mi primera experiencia laboral y ha sido inmejorable. Gracias a todos por crear un ambiente de trabajo tan magnífico.

Por último, me gustaría agradecer su apoyo y guía a mi tutor, Carlos, sin el que no hubiese sido posible estre proyecto. Gracias por brindarme la oportunidad de trabajar en el GSI y poder llevar a cabo este proyecto.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This chapters provides a main introduction to the problem approached in this project. It describes an overview of the benefits of linked data technologies and its application to geographical area. We analyse the state of the art and present the main purpose of this project. Finally, a deeper description of the project and its context is also given.

1.1 Context

In recent years, the amount of information available on the web has grown exponentially. This has promoted an evolution of the web as we know so the data published on the net was in such a way that it is machine-readable by adding semantic metadata to the traditional data. This has led to the Semantic Web and Linked Data.

The principles of Linked Data were first outlined by Berners-Lee in 2006 [1]:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Include links to other URIs. so that they can discover more things.

The Semantic Web is a set of activities performed within World Wide Web Consortium (W3C[1]) whose purpose is to create technologies to publish machine-readable data applications. Thus, the concept of Semantic Web involves an expansion of the traditional Web, where semantic metadata is added to the traditional data on the web. This metadata describes the content, meaning and the relationships between the available data. The metadata is formally provided as standard, so computers can understand and process themselves these new data automatically. In this way we extend the interoperability between different software agents.

In the geospatial context, GeoLinked Data[2] in an open initiative whose aim is to enrich the Semantic Web with geospatial data into the context of INSPIRE[3] *(INfrastructure for SPatial InfoRmation in Europe)* Directive. This initiative focuses its efforts to collect, process and publish geographic information from different organizations around the world and providing the suitable tools for handing all the data.

Once the information has been published on the web, we need visualization tools that allow us to query the different datasets available online and visualize the retrieved data. With these tools we will take the maximum advantage of the semantic web and all its benefits.

[1]http://www.w3c.es/
[2]http://linkedgeodata.org/
[3]http://inspire.ec.europa.eu/

1.2 Master thesis goals

The main goal of this Master Thesis is to develop a web application that allows us to query any dataset and manage, filter and visualize the retrieved data. The different challenges to achieve this goal are:

- Analyse the state of the Semantic Web technologies and study all related standards (RDF, OWL, SPARQL...)
- Study the different web technologies that will help us to develop the application. In particular, we will study the main application used for the project: Sefarad.
- Determine the architecture of the application.
- Develop each one of the modules that make up the system.
- Test the application on different case studies to determine possible bugs, possible improvements or guarantee proper operation.
- Document the work done for future users and developers.

1.3 Structure of this Master Thesis

In this section we will provide a brief overview of all the chapters of this Master Thesis. It has been structured as follows:

Chapter 1 provides an introduction to the problem which will be approached in this project. It provides an overview of the benefits of linked data technologies. Furthermore, a deeper description of the project and its environment is also given.

Chapter 2 contains an overview of the existing technologies on which the development of the project will rely.

Chapter 3 describes one of the most important stages in software development: the requirement analysis using different scenarios. For this, a detailed analysis of the possible use cases is made using the Unified Modeling Language (UML). This language allows us to specify, build and document a system using graphic language. The result of this evaluation will be a complete specification of the requirements, which will be matched by each module in the design stage. This helps us also to focus on key aspects and take apart other less important functionalities that could be implemented in future works.

Chapter 4 describes the architecture of the system, divided in several modules with its own purpose and functions.

Chapter 5 describes a selected use cases. It is going to be explained the running of all the tools involved and its purpose. It allows us to test the application and give us some feedback to improve our system and repair bugs and errors.

Chapter 6 sums up the findings and conclusions found throughout the document and gives a hint about future development to continue the work done for this master thesis.

Finally, the appendix provides useful related information, especially covering the installation and configuration of the tools used in this thesis.

CHAPTER 2

Enabling Technologies

This chapter introduces which technologies have made possible this project. First of all we must introduce Linked Data and, specifically, GeoLinked Data [2, 3] and all its possibilities.Then we present Sefarad[1], an HTML5 Framework developed by Grupo de Sistemas Inteligentes (GSI[2]) which provides us a semantic front end to Linked Data (LOD) [4]. Finally, we introduce the other technologies that have helped us to develop this project.

[1]https://github.com/gsi-upm/Sefarad
[2]http://www.gsi.dit.upm.es/

2.1 Overview

Linked Data is a technological innovation that transforms the way we think about information and its role in society, in our case geographic information. Linked Data has been recently suggested as one of the best alternatives for creating these shared information spaces [5]. Linked Data describes a method of publishing structured and related data so that it can be interlinked and become more useful, which results in the Semantic Web[3] (or Web of Data). It builds upon standard Web technologies such as HTTP, RDF and URIs, but rather than using them to serve web pages for human readers, it extends them to share information in a way that can be read automatically by computers. This enables data from different sources to be connected and queried using SPARQL standard. This is specially important for sophisticated types of information, in particular information with spatial and temporal components.

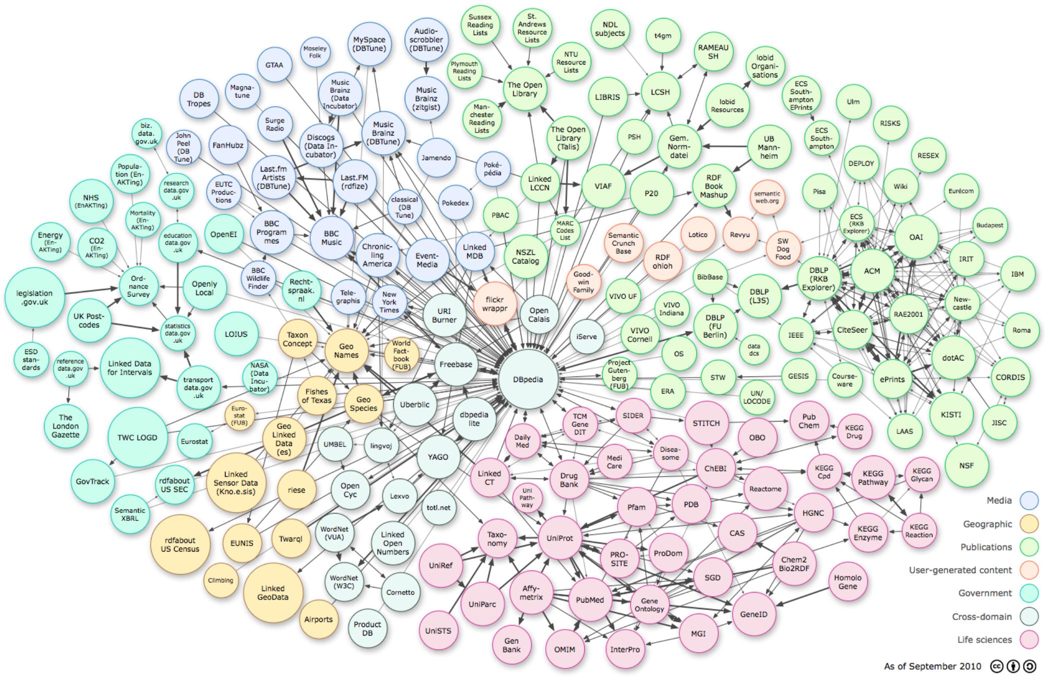


Figure 2.1: Linked Open Data cloud diagram example[4]

With the adoption of Linked Data, the traditional complexities of conceptual database schemata for spatial data can safely remain internal to organizations. Their externally relevant contents get streamlined into the open and more manageable form of vocabulary definitions. Users of Linked Data do not need to be aware of complex schema informa-

[3]http://www.w3.org/standards/semanticweb/
[4]http://lod-cloud.net/

tion to use data adequately, but ”only” of the semantics of types and predicates (such as isLocatedIn) occurring in the data. While many questions remain to be answered about how to produce and maintain vocabulary specifications, the elaborate layering of syntactic, schematic, and semantic interoperability issues has simplified to a single common syntax (RDF[5]), the irrelevance of traditional schema information outside a database, and a focus on specifying and sharing vocabularies.

This simplification is more dramatic for spatially and temporally referenced data (with their complexities in the form of geometries and scale hierarchies). The resulting paradigm shift, from distributed complex databases accessed through web services that expose schemata to knowledge represented as graphs, whose links can be given well-defined meaning, radically changes some of the long-standing problems of GIScience and GIS practice. Everything said is a way to facilitate analysis and integration of all geographic information available worldwide.

This master thesis describes the creation of a web application for GeoLinked Data management and visualization. The main goal of this project is to develop a web application that facilitates the handling and visualization of GeoLinked Data available worldwide. Users could manage geographical information from any SPARQL endpoint just running their own queries or graph their own datasets storing them in a semantic web server such as Linked Media Framework (LMF[6]) or in a NoSQL database such as MongoDB[7].

2.2 Linked Data in a Nutshell

The rise of the Open Data Movement has led to the Web of Data grow significantly over the last years. This Web of Data has started to span data sources form a wide range of domains such as people, companies, music, scientific publications, etc. The principles of Linked Data were first outlined by Berners-Lee in 2006 [1]:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Include links to other URIs. so that they can discover more things.

[5]http://www.w3.org/RDF/
[6]https://code.google.com/p/lmf/
[7]http://www.mongodb.org/

Linked Data is the name for a collection of design principles, practices and technologies centered around a novel paradigm to expose, publish, retrieve, reuse, and integrate data on the Web. In summary, that is simply about using the Web to create typed links between data from different sources. In contrast to the Document Web, the Semantic Web aims at establishing named and directed links between typed data. For example, a normal Web page about Portsmouth (such as http://en.wikipedia.org/wiki/Portsmouth) may link to another page about Hampshire (such as http://en.wikipedia.org/wiki/Hampshire). For a machine, the intended meaning of such links is difficult to interpret and the Web pages can only be consumed as integral units of text or other media. On the Linked Data Web, by contrast, the link between Portsmouth and Hampshire would be directed and labelled, for example, forming the statement that Portsmouth is located in Hampshire. Additionally, the two places would be typed, e.g., as city and county, jointly leading to the statement that the city of Portsmouth is located in the county of Hampshire. Finally, the predicate isLocatedIn could be defined as a transitive relation in an ontology. Thus, in conjunction with a statement that Hampshire county is located in the UK, one could automatically derive the new statement that Portsmouth is located in the UK.

```xml
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#" xmlns:eric="http://www.
    w3.org/People/EM/contact#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:mailbox rdf:resource="mailto:e.miller123(at)example"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:personalTitle>Dr.</contact:personalTitle>
  </rdf:Description>
</rdf:RDF>
```

Listing 2.1: RDF/XML document example

Technically, Linked Data refers to data published on the Web in such a way that it is machine-readable, its meaning is explicitly defined, it is linked to other external data sets, and can in turn be linked to from external data sets. That three given elements constitute each piece of information in Linked Data, one refers to such statements as triples, consisting of a subject (Portsmouth), a predicate (isLocatedIn), and an object (Hampshire). This syntax, which happens to be the simplest form in which statements can be made in natural language, has thus been carried over to the world of data. The data model for triples is the so-called Resource Description Framework. Every entity in the physical world (even a subject, a predicate or an object) should be identified by a global unique URI, and all the information should be provided by using W3C standards such as mentioned RDF or OWL.

Linked Data can be queried using SPARQL[8] (an acronym for SPARQL Protocol and RDF Query Language), a query language for RDF which became an official W3C Recommendation[9]. The SPARQL query language consists of the syntax and semantics for asking and answering queries against RDF graphs and contains capabilities for querying by triple patterns, conjunctions, disjunctions, and optional patterns. It also supports constraining queries by source RDF graph and extensible value testing. Results of SPARQL queries can be ordered, limited and offset in number, and presented in several different forms, such as JSON, RDF/XML, etc.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

Listing 2.2: SPARQL query example

In our particular case of dealing with geographic information, GeoSPARQL[10] enriches SPARQL by quantitative reasoning. Linked Data is usually stored and accessed via SPARQL endpoints (e.g., DBpedia[11] or GeoNames[12]). The ontologies that allow human users and machines to understand which concepts and predicates can be queried, and how they are formally defined, are described using languages such as the Web Ontology Language (OWL[13]).

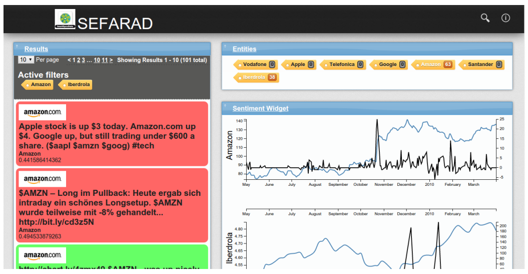
In the geospatial context, GeoLinked Data[14] is an open initiative whose aim is to enrich the Semantic Web with geospatial data into the context of INSPIRE[15] *(INfrastructure for SPatial InfoRmation in Europe)* Directive. This initiative focuses its efforts to collect, process and publish geographic information from different organizations around the world and providing the suitable tools for handing all the data.

[8]http://www.w3.org/TR/rdf-sparql-query
[9]http://www.w3.org/blog/SW/2008/01/15/
[10]http://www.opengeospatial.org/standards/geosparql
[11]http://dbpedia.org/
[12]http://www.geonames.org/
[13]http://www.w3.org/2001/sw/wiki/OWL
[14]http://linkedgeodata.org/
[15]http://inspire.ec.europa.eu/

2.3 Sefarad

Sefarad is a web application developed to explore linked data by making SPARQL queries to the chosen endpoint without writing more code, so it provides a semantic front-end to Linked Open Data [4]. It allows the user to configure his own dashboard with many different widgets to visualize, explore and analyse graphically the different characteristics, attributes and relationships of the queried data. Sefarad is developed in HTML5[16] and follows a Model View-View Model (MVVM) pattern performed with the Knockout[17] framework. This JavaScript library allows us to create responsive and dynamic interfaces which automatically is updated when the data changes. The different parts of the UI are connected to the data model by declarative bindings.

Sefarad consists of two different tabs: dashboard and control panel. The first tab allows the user to perform faceted search on the data accessed, so the users can explore a collection of information by applying multiple filters. In the control panel tab statistics about the dataset are visualized.



amazon.com
Apple stock is up $3 today. Amazon.com up $4. Google up, but still trading under $600 a share. ($aapl $amzn $goog) #tech
Amazon
0.441586414362

amazon.com
$AMZN – Long im Pullback: Heute ergab sich intraday ein schönes Longsetup. $AMZN wurde teilweise mit -8% gehandelt... http://bit.ly/cd3z5N
Amazon
0.494533879263

amazon.com

Figure 2.2: Main Layout

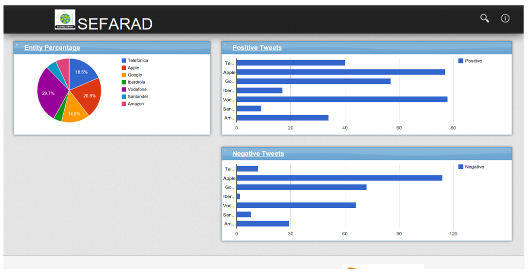


Figure 2.3: Dashboard

The great potential of Sefarad for our project lies in the capability to create our own widgets really easily. We should not worry about obtaining the filtered data and updating the widget when a new facet is selected thanks to Knockout framework. For this purpose, the application specifies how to create a new Javascript file in which it should be placed a Javascript object using D3.js[18] framework. We will take advantage of this feature to develop geographic widgets. The widget template is as follows:

[16]http://www.w3.org/TR/html5/
[17]http://knockoutjs.com/
[18]http://d3js.org/

```
// New widget
var newWidget = {
    // Widget name.
    name: "Name",
    // Widget description.
    description: "description",
    // Path to the image of the widget.
    img: "path/to/image",
    // Type of the widget.
    type: "type",
    // Help display on the widget
    help: "help",
    // Category of the widget (1: textFilter, 2: numericFilter, 3: graph,
        5:results, 4: other, 6:map)
    cat: X,

    render: function() {
        var id = 'A' + Math.floor(Math.random() * 10001);
        var field = newWidget.field || "";
        vm.activeWidgetsRight.push({
            "id": ko.observable(id),
            "title": ko.observable(newWidget.name),
            "type": ko.observable(newWidget.type),
            "field": ko.observable(field),
            "collapsed": ko.observable(false),
            "showWidgetHelp": ko.observable(false),
            "help": ko.observable(newWidget.help)
        });
        newWidget.paint(id);
    },

    paint: function(id) {
        d3.select('#' + id).selectAll('div').remove();
        var div = d3.select('#' + id);
        div.attr("align", "center");

        // Code to paint
    }
};
```

Listing 2.3: "Sefarad widget template"

2.4 MongoDB: a NoSQL Database

MongoDB[19] is an open-source document-oriented NoSQL database distributed under the GNU Affero General Public License[20] and the Apache License[21], written in the programming language C. As a NoSQL database, instead of traditional table-based relational database MongoDB is structured into collections. Those collections are a set of BSON (Binary JSON) documents containing a set of fields or key-value pairs: keys are string and value cans be of so many types (string, integer, float, timestamp, etc.). That provides high performance, high availability, and automatic scaling. Figure [2.4] shows the possible similarities or equivalences between MongoDB and traditional relational databases.

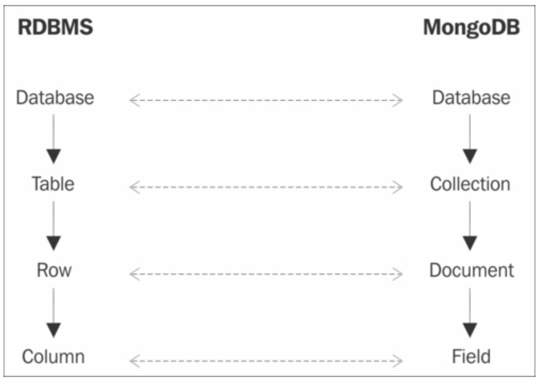


Figure 2.4: Comparison between relational and MongoDB data models[22]

In MongoDB, the basic piece of data is called a document[2.5]. A major advantage in MongoDB is that documents do not have a predefined schema (*flexible schema*). We can think of a document as a multidimensional array whose values could themselves be another array. In practical matters, MongoDB documents have a JSON array structure.

Furthermore, MongoDB is optimized for CRUD operations. You can store as much information as you need in a document without first defining its structure, and this data will be able to be queried. In order to retrieve one o more documents, you may run your own query specifying some criteria or conditions. A query may support search by field, range or conditional statements such as the existence or not of a key. This make the system highly scalable.

[19]https://www.mongodb.org/
[20]http://www.gnu.org/licenses/agpl-3.0.html
[21]http://www.apache.org/licenses/LICENSE-2.0.html

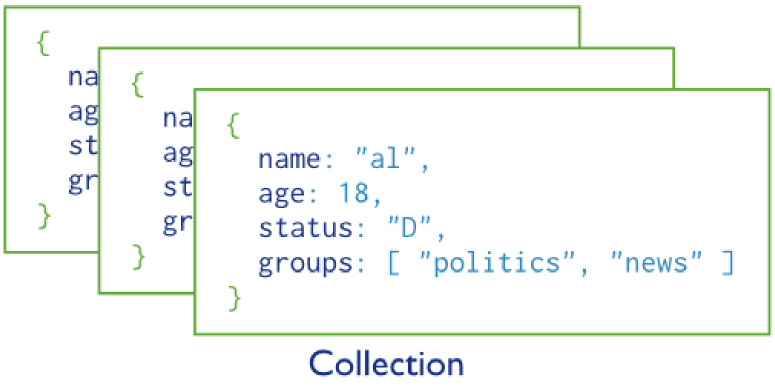


Figure 2.5: A collection of MongoDB documents[23]

MongoDB also offers the possibility of replication into two or more replica sets, providing high availability. Every moment one of the replica sets works as the primary and replaces and updates the data of the replicas. When the primary fails, the secondary replica becomes principal. Additionally, MongoDB can run simultaneously over multiple servers, balancing the load between them and keeping those security replica sets and the system running in case of hardware failure.

MongoDB supports drivers for most common programming languages. Due to the fact that the structure of a document is similar to a JSON object and most of programming languages drivers support the management and conversion of JSON datatypes to language-specific structures, it is too easy to communicate and manipulate the data. In the case of this project, we use the PHP[24] driver for MongoDB.

2.5 GeoServer

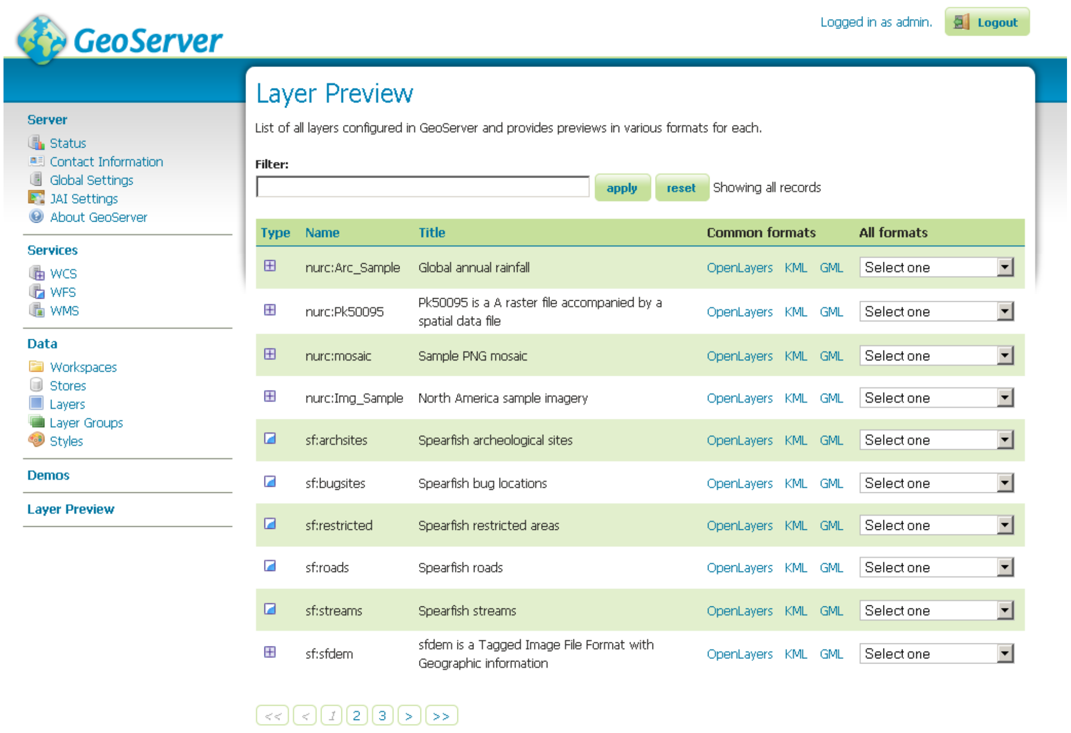
GeoServer[25] [6] is an open source software server written in Java that allows users to view and edit geospatial data. GeoServer functions as the reference implementation of the Open Geospatial Consortium Web Feature Service[26] standard, and also implements the Web Map Service[27], Web Coverage Service[28] and Web Processing Service[29] specifications.

Some of the main features of GeoServer are:

[23]http://docs.mongodb.org/manual/core/crud-introduction/
[24]http://php.net/manual/es/book.mongo.php
[25]http://geoserver.org/
[26]http://www.opengeospatial.org/standards/wfs
[27]http://www.opengeospatial.org/standards/wms
[28]http://www.opengeospatial.org/standards/wcs
[29]http://www.opengeospatial.org/standards/wps

- Full compatible with the OGC specifications.
- Easy installation and configuration (no large configuration files needed)
- Multiple formats supported, such as PostGIS or Shapefile.
- Multiple output formats supported, such as JPEG, GIF, PNG, SVG y GML.
- ECQL query language support.

GeoServer also includes an administration UI[2.6] from which users can manage the stored data, observe and analyse the different attributes of the information as well as a preview of the different layers for what GeoServer includes an integrated OpenLayers client. In our project we save our geospatial dataset into a GeoServer installation and display the information with maps integrated into Sefarad as a new widget developed with OpenLayers.js.



Layer Preview

List of all layers configured in GeoServer and provides previews in various formats for each.

Showing all records

Type	Name	Title	Common formats	All formats
⊞	nurc:Arc_Sample	Global annual rainfall	OpenLayers KML GML	Select one
⊞	nurc:Pk50095	Pk50095 is a A raster file accompanied by a spatial data file	OpenLayers KML GML	Select one
⊞	nurc:mosaic	Sample PNG mosaic	OpenLayers KML GML	Select one
⊞	nurc:Img_Sample	North America sample imagery	OpenLayers KML GML	Select one
◩	sf:archsites	Spearfish archeological sites	OpenLayers KML GML	Select one
◩	sf:bugsites	Spearfish bug locations	OpenLayers KML GML	Select one
◩	sf:restricted	Spearfish restricted areas	OpenLayers KML GML	Select one
◩	sf:roads	Spearfish roads	OpenLayers KML GML	Select one
◩	sf:streams	Spearfish streams	OpenLayers KML GML	Select one
⊞	sf:sfdem	sfdem is a Tagged Image File Format with Geographic information	OpenLayers KML GML	Select one

Figure 2.6: Geoserver administration UI

2.6 Fuseki: RDF over HTTP server

Fuseki[30] is a SPARQL server. It provides REST-style SPARQL HTTP Update, SPARQL Query, and SPARQL Update using the SPARQL protocol over HTTP. We will use a Fuseki server installation for storing our own RDF files containing geoSPARQL data.

2.7 OpenLayers

OpenLayers[31] [7][8] is the most complete and powerful open source JavaScript library to create any kind of web mapping application. OpenLayers was originally developed and released by MetaCarta under a BSD license.

In addition to offering a great set of components, such as maps, layers, or controls, OpenLayers offers access to a great number of data sources using many different data formats, implements many standards from Open Geospatial Consortium (OGC), and is compatible with almost all browsers. Because of this, OpenLayers allows us to display the geographical information stored in all major and common data servers into functional and interactive maps. This means the users can connect your client application to web services spread, add data from a bunch of raster and vector file formats such as GeoJSON and GML, and organize them in layers to create original web mapping applications.

OpenLayers provides lots of controls such as pan, zoom, and query the map to build interactive maps which give users the possibility to actually explore the maps and the geospatial data display on them. OpenLayers allows you to include as many layers as you want, each representing a piece of information. Each layer can be customized with different colours, transparency, shadows, alive and clicking information, etc., and can be shown or hidden every moment. There are two kinds of layers in OpenLayers: base and non-base. Base layers are always visible and determines some of the essential properties of the map (zoom, center, etc.). A map can have more than one base layer but only one of them can be active at a time.

In the case of GeoLinked Data, OpenLayers provides us the necessary tools to represent geographical information stored in GeoJSON[32] in a map. GeoJSON is a format for encoding a variety of geographic data structures and supports multiple geometry types, such as *Point, LineString, Polygon, MultiPoint, MultiLineString, and MultiPolygon.*

[30]http://jena.apache.org/documentation/serving_data/
[31]http://openlayers.org/
[32]http://geojson.org/

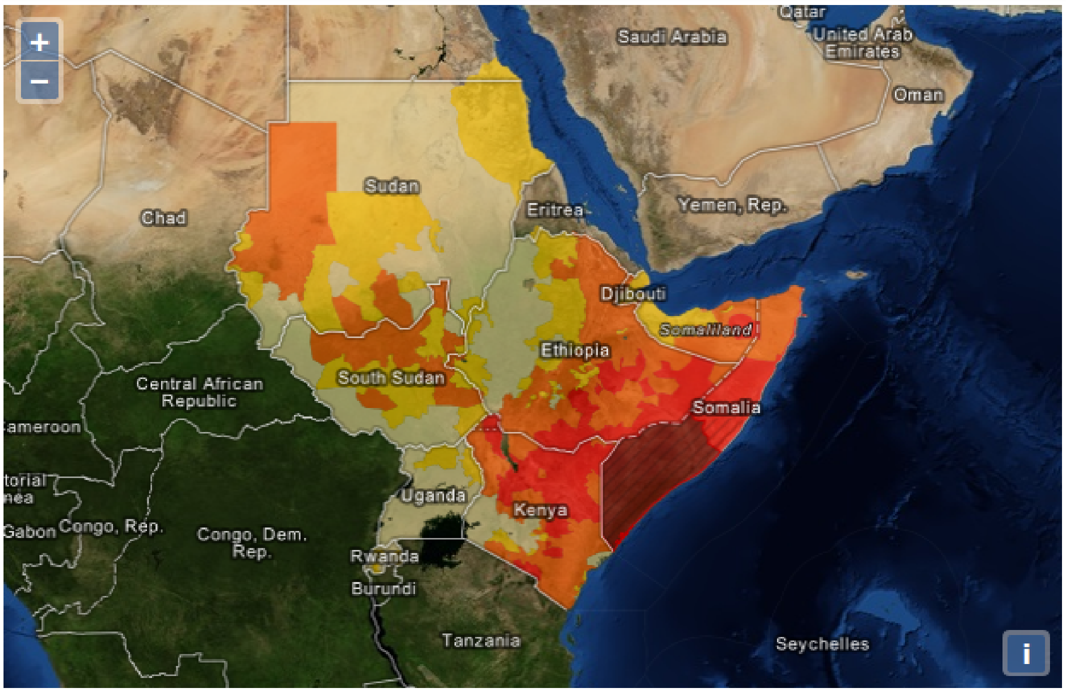


Figure 2.7: OpenLayers map example with multiple layers

2.8 Grunt: The JavaScript Task Runner

GruntJS[33] [9] is a JavaScript task runner written with Node.js[34] used to automate predefined tasks to ease the development and integration of our project and to save time automating repetitive tasks.

Grunt provides lots of plugins that are installed and manage via npm[35], Node.js package manager, which allows us to automate some manual repetitive tasks we run as part of our development or deployment process. Those plugins are labelled *contrib* packages, which means they are branded as officially maintained and stable. Moreover, users share their own plugins and every one can easily create their own user-defined task plugin if there is no one for the task they want to automate.

To automate your project with Grunt, you must implement your Gruntfile.js configuration file and a package.json file. In the configuration file we indicate the tasks we want to automate and load the corresponding plugins with a simple command and configure them with JSON format. The package.json file is used to list grunt and the Grunt plugins your project needs as npm *devDependencies*.

[33]http://gruntjs.com/
[34]http://nodejs.org/
[35]https://www.npmjs.org/

Once the two mentioned files have been created, each time grunt is run it looks for a locally installed Grunt. When it is found, the CLI (Grunt's Command Line Interface) loads the local installation of the Grunt library, applies the configuration from your **Gruntfile**, and executes any tasks you've requested for it to run.

CHAPTER 3

Requirement Analysis

This chapter describes one of the most important stages in software development: the requirement analysis using different scenarios. For this, a detailed analysis of the possible use cases is made using the Unified Modeling Language (UML). This language allows us to specify, build and document a system using graphic language.

3.1 Overview

The result of this chapter will be a complete specification of the requirements, which will be matched by each module in the design stage. This helps us also to focus on key aspects and take apart other less important functionalities that could be implemented in future works.

3.2 Actors dictionary

The list of primary and secondary actors is presented in table [3.1]. These actors participate in the different use cases, which are presented later.

Actor identifier	**Role**	**Description**
ACT-1	Portal User	End user without technical knowledge on Semantic Technologies that uses Sefarad to query a SPARQL endpoint, display the retrieved information and use the faceted search. These users need an intuitive and clear interface and an appropriate help section.
ACT-2	Advanced User	End user with technical knowledge on Semantic Technologies. Theses users can edit their own SPARQL queries and use more complex configurations because of their knowledge.
ACT-3	Admin	Administrator of Sefarad, in charge of tasks such as security management, inserting and deleting datasets in local database, etc.

Table 3.1: Actors list

3.3 Use cases

This section identifies the use cases of the system. This helps us to obtain a complete specification of the uses of the system, and therefore define the complete list of requisites to match. First, we will present a list of the actors in the system and a UML diagram (Figure 3.1) representing all the actors participating in the different use cases. This representation allows us to specify the actors that interact in the system and the relationships between them.

These use cases will be described the next sections, including each one a table with their complete specification. Using these tables, we will be able to define the requirements to be established.

The next graphic represents all the use cases involved in this project in a UML diagram.

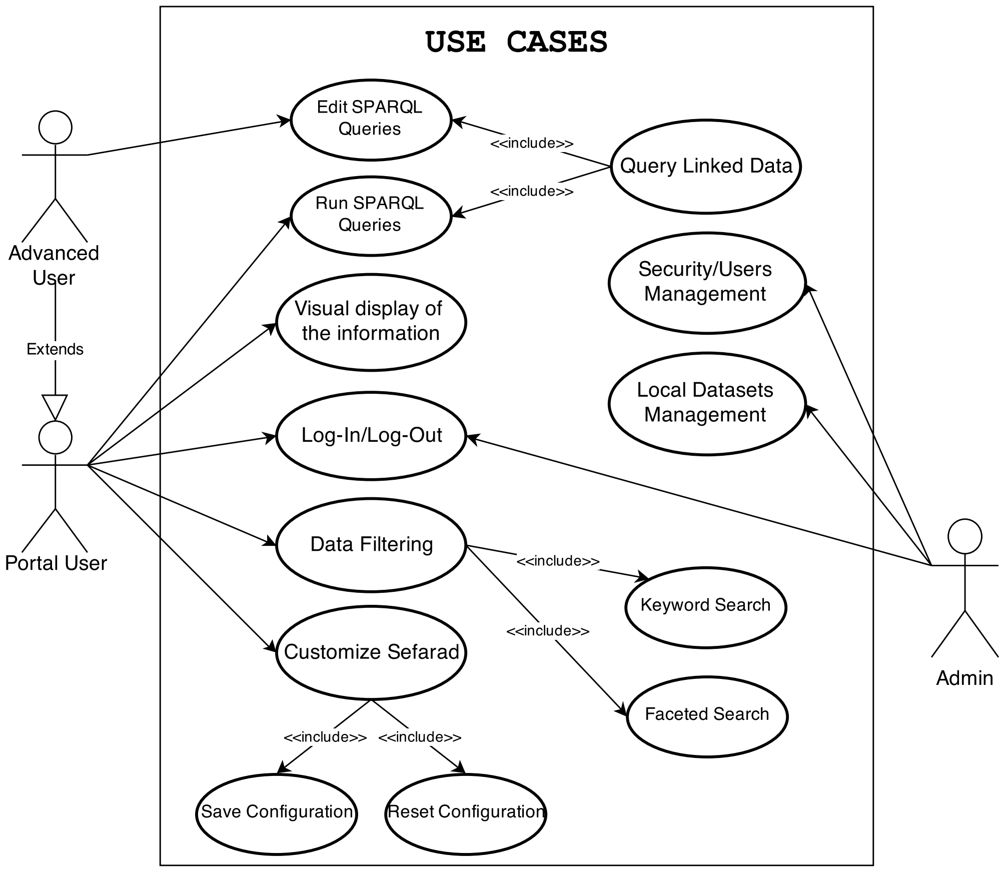


Figure 3.1: Use cases UML diagram

3.3.1 Portal users use cases

The use cases presented in this section are those related to all the portal users. These are:

- *Edit SPARQL queries* detailed in (Section 3.3.1.1).
- *Run SPARQL queries* detailed in (Ssection 3.3.1.2).
- *Visual display of the information* detailed in (Section 3.3.1.3).
- *Keyword search* detailed in (Section 3.3.1.4).
- *Faceted search* detailed in (Section 3.3.1.5).
- *Log-in/Log-out* detailed in sub-section 3.3.1.6).
- *Customize Sefarad* detailed in (Section 3.3.1.7).
- *Save own configuration* detailed in (Section 3.3.1.8).
- *Reset own configuration* detailed in (Section 3.3.1.9).

3.3.1.1 Edit a SPARQL query

Use Case Name	Edit SPARQL query		
Use Case ID	UC1.1		
Primary Actor	Advanced User		
Flow of Events		Actor Input	System Response
	1	The user selects the SPARQL Editor to write his own query	A new text-box is opened for the user to write the query
	2	The user writes the query	The SPARQL Editor assists the user by displaying different valid options for the query or highlighting errors
	3	The user saves the query	The new query is saved for later execution

Table 3.2: Edit a SPARQL query

3.3.1.2 Run a SPARQL query

Use Case Name	Run SPARQL query		
Use Case ID	UC1.2		
Primary Actor	Portal User		
Pre-Condition	The user has selected or edit a SPARQL query		
Flow of Events		Actor Input	System Response
	1	The user selects a SPARQL endpoint	The configuration of the application is updated so the following queries will be run against the selected endpoint
	2.1	The user executes the query	The SPARQL query is executed against the selected endpoint
	2.2		The server responds with results data. The data available is updated with the information retrieved and all the widgets are automatically updated

Table 3.3: Run a SPARQL query

3.3.1.3 Visual display of the information

Use Case Name	Visual display of the information		
Use Case ID	UC1.3		
Primary Actor	Portal User		
Pre-Condition	The application has received response data from a SPARQL query		
Flow of Events		Actor Input	System Response
	1	The user adds a new widget selecting which facet or facets to show	A widget is deployed configured with the selected facet or facets. This widget is automatically updated when the filtered data is updated

Table 3.4: Visual display of the information

3.3.1.4 Keyword search

Use Case Name	Keyword Search		
Use Case ID	UC1.4		
Primary Actor	Portal User		
Pre-Condition	The application has received response data from a SPARQL query and it has been indexed		
Flow of Events		Actor Input	System Response
	1	The user expresses her goals by typing textual keywords into the search box	The keywords entered are stored as a new filter
	2		The filtered data available in the application is updated by using the new filters selected by the user to filter on any of the fields of the data
	3		The widget layout is automatically updated

Table 3.5: Keyword search

3.3.1.5 Faceted search

Use Case Name	Faceted Search		
Use Case ID	UC1.5		
Primary Actor	Portal End-User		
Pre-Condition	The application has received response data from a SPARQL query and it has been indexed		
Flow of Events		Actor Input	System Response
	1	The user adds a new faceted search widget selecting which facet to filter	A new faceted widget is deployed configured with the selected facet. This widget shows all the possible values for this facet
	2	The user selects a new values for this facet to filter by	The filtered data is automatically updated with the new criteria
	3		The widget layout is automatically updated

Table 3.6: Faceted search

3.3.1.6 Log-in/Log-out

Use Case Name	Log-in/Log-out		
Use Case ID	UC1.6		
Primary Actor	Portal User & Admin		
Pre-Condition	The application has been initialized without any user logged in		
Post-Condition	The user can log out any moment		
Flow of Events		Actor Input	System Response
	1	The user introduces his credentials (username and password)	On the server-side the application finds matches for the username/password introduced in MongoDB users database
	2.a	The user introduces valid credentials	The user is authenticated. The interface of the application is updated to the admin interface, showing hidden buttons such as 'add new widget', 'delete widget', 'save configuration', etc.
	2.b	The user introduces no valid credentials	An alert message reports that authentication has failed.
	3	The user clicks 'logout' button	The session is closed and the application turns back to no-admin interface

Table 3.7: Log-in/Log-out

3.3.1.7 Customize Sefarad

Use Case Name	Customize Sefarad		
Use Case ID	UC1.7		
Primary Actor	Portal User		
Pre-Condition	The application has been initialized and the user has logged in		
Flow of Events		Actor Input	System Response
	1.1	The user adds a new widget	The new widget is stored and displayed in the layout
	1.2	The user removes a widget	The widget is deleted and removed from the layout
	1.3	The user configures a widget	The new configuration is stored and the widget is redrawn
	1.4	The user reorders the widget layout	The new configuration is stored
	1.5	The user changes the global configuration	The new configuration is stored

Table 3.8: Customize Sefarad

3.3.1.8 Save own configuration

Use Case Name	Save configuration		
Use Case ID	UC1.8		
Primary Actor	Portal User		
Pre-Condition	The application has been initialized and the user has logged in		
Flow of Events		Actor Input	System Response
	1	The user clicks 'save configuration' button	The old saved configuration for the user is removed from the database
	2		The actual configuration is inserted into the database

Table 3.9: Save own configuration

3.3.1.9 Reset own configuration

Use Case Name	Reset configuration		
Use Case ID	UC1.9		
Primary Actor	Portal User		
Pre-Condition	The application has been initialized and the user has logged in and the user has a configuration saved		
Flow of Events		Actor Input	System Response
	1	The user clicks 'reset configuration' button	The old saved configuration for the user is removed from the database
	2		The default configuration is retrieved from the database
	3		The actual configuration is replaced with the default configuration retrieved

Table 3.10: Reset own configuration

3.3.2 Admin use cases

This use case package collects the use cases related to admin users. The use cases presented in this section are:

- *Security/Users Management* detailed in (Section 3.3.2.1).
- *Local Datasets Management* detailed in (Section 3.3.2.2).

3.3.2.1 Security and users management

Use Case Name	Security/Users Management		
Use Case ID	UC2.1		
Primary Actor	Admin		
Pre-Condition	The administrator of the system has logged-in as admin into the MongoDB database		
Flow of Events		Actor Input	System Response
	1.1	The admin inserts a new user into the database	The new user document is stored in the users collection
	1.2	The admin removes a user from the database	The user document is deleted from the users collection
	1.3	The admin edits user permissions or information	The user information is updated

Table 3.11: Security and users management

3.3.2.2 Local datasets management

Use Case Name	Local datasets management		
Use Case ID	UC2.2		
Primary Actor	Admin		
Pre-Condition	The administrator of the system has logged-in as admin into the local database		
Flow of Events		Actor Input	System Response
	1.1	The admin inserts data into the local dataset	The new data is stored
	1.2	The admin removes data from the dataset	The data is removed
	1.3	The admin updates the local dataset	The dataset is updated and saved

Table 3.12: Local datasets management

3.3.3 Conclusions

With the use cases described we have introduced the basic functionalities that have been implemented in this project. They help us to understand the different actors that can interact. They can serve as a base for further development and different use cases that can come to mind.

CHAPTER 4

Architecture

This chapter describes in depth how the system is structured in different modules and how the users interact with them. We will describe each one of these modules describing its main purpose, structure and function. After reading this chapter, the user will know how the application and each of its modules work.

4.1 Introduction

In this chapter we show a detailed diagram [4.1] about the complete architecture of Sefarad. In the first section we introduce that scheme and the behaviour and the main function of each of the modules and components. After this, in the following subsections we describe each module in depth showing specific diagrams, screenshots and detailing their particular operation.

4.2 Architecture

A diagram of the architecture is shown in Figure 4.1. Each module is detailed in the following sections.

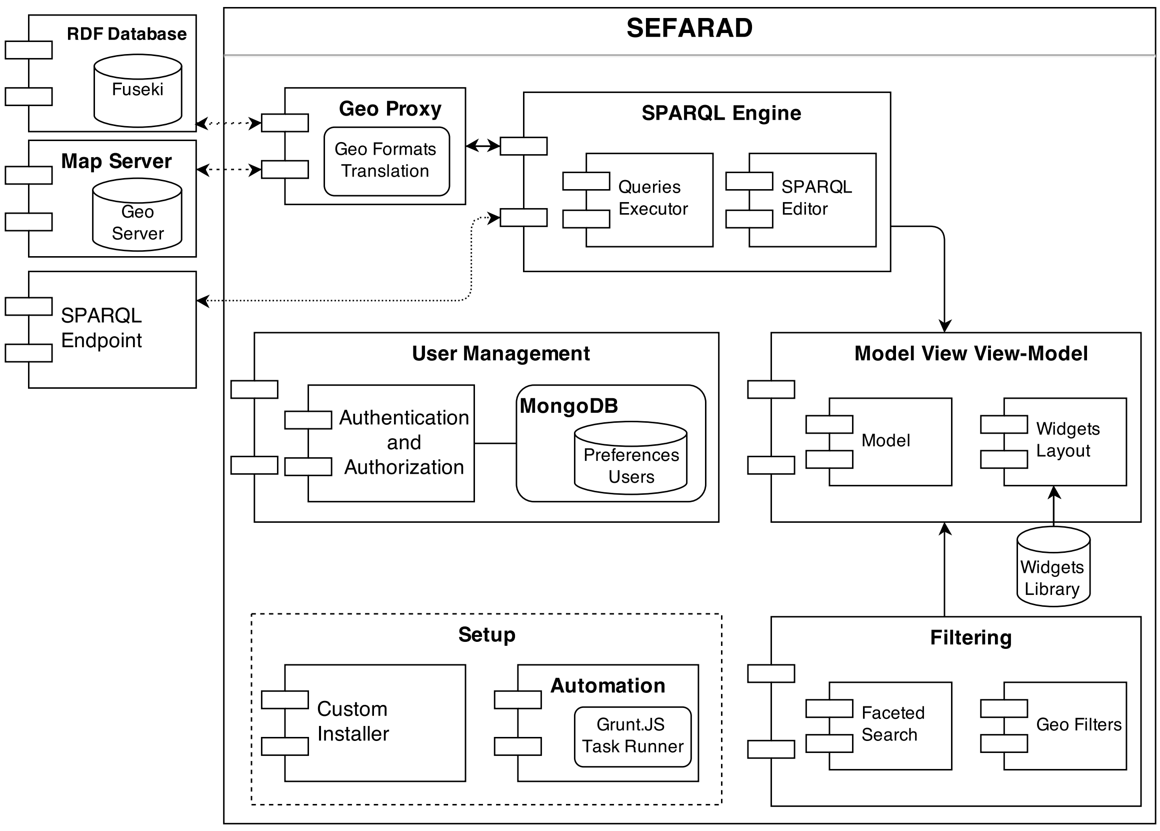


Figure 4.1: General Architecture

Since the main purpose of this master thesis is to develop a HTML5 Framework to query, manage and represent *Geo Linked Data*, we need a *SPARQL Engine* to edit and execute the queries to the endpoint we want and retrieve the data. Non-technical users can edit their queries by using a SPARQL editor which will help them with recommendations and corrections while editing the query and advanced users can edit their own queries. The application allows the users to query any semantic repository with a corresponding SPARQL endpoint or any local dataset within a local database such as *Fuseki and Virtuoso* or *Geo Server*. In order to get the data in a proper format for Sefarad, the *Geo Proxy module* handles the conversion of the data when it is needed.

Once the data is retrieved, the *Search and filtering module* provides us the necessary tools to manage the queried data enabling *Faceted search*, *Keyword search* and *Geo filtering*. The application automatically indexes, sorts and classifies the information, obtaining the different facets and values of the data. All the filtering services are provided to the user in an intuitive graphical interface by providing multiple widgets, a search box and a sortable table of results.

To handle the changes in the filtered data due to the different filters selected by the user, the *Model View View-Model module* uses the features offered by Knockout framework. Every time a new search or filter criteria is included, the data model will be automatically updated with the results that meet the new conditions and all the widgets displayed in the layout will be redrawn using updated final results.

For the security and administration tasks, we need a *User management module*. This module includes a *Security: authentication and authorization sub-module* based on PHP and MongoDB. The user's username and password (encoded in MD5 hash) are stored in a MongoDB collection named *users*. When a user wants to log in, the application checks his user credentials with a PHP5 script. In case of success, depending on the user's permissions (admin, basic user, etc.), the different tools and options are shown or hidden (i.e. add, configure and delete widgets). Furthermore, the users preferences and settings are stored in another MongoDB collection, so that when a user logs in the application is configured using the last configuration saved by the user. This is managed by *MongoDB: settings and preferences sub-module*.

Finally, the *Setup module* provides a graphical installer for an easy deployment of the application in any computer running an Linux operating system, installing anything needed to run Sefarad. This module includes two sub-modules: a *Custom installer*, which allows the user to select which modules to include in its installation; and an *automation module*, to automate certain repetitive tasks with a single command.

4.3 SPARQL Engine

The SPARQL engine is the main module for managing the SPARQL queries. It provides the necessary tools for editing and running our own queries to the selected dataset, what is the first step in the indexing and processing of the information requested.

Sefarad allows the user to query different kinds of datasets. The user can query a local dataset stored into a geographic server such as *GeoServer* or a local RDF database such as *Fuseki* or *Virtuoso*. The user can also select a SPARQL endpoint provided by any website to query their public datasets. All these options can be configured in the SPARQL tab in the configuration window. After selecting the endpoint, the user can write his own query and execute it. Data retrieved in response to the query will be stored in the application as a JSON object. This data will be managed by both the *Search and filtering module* and the *Model View View-Model* in order to allow all the filtering features, the display and the update of the final results and widgets.

The SPARQL Engine module consists of two submodules: *SPARQL Editor*, which supports the user to write his own query by showing errors in the query language or proposing valid options; and the *SPARQL queries executor*, which allows the user to run the query and retrieve the data.

4.3.1 SPARQL Editor

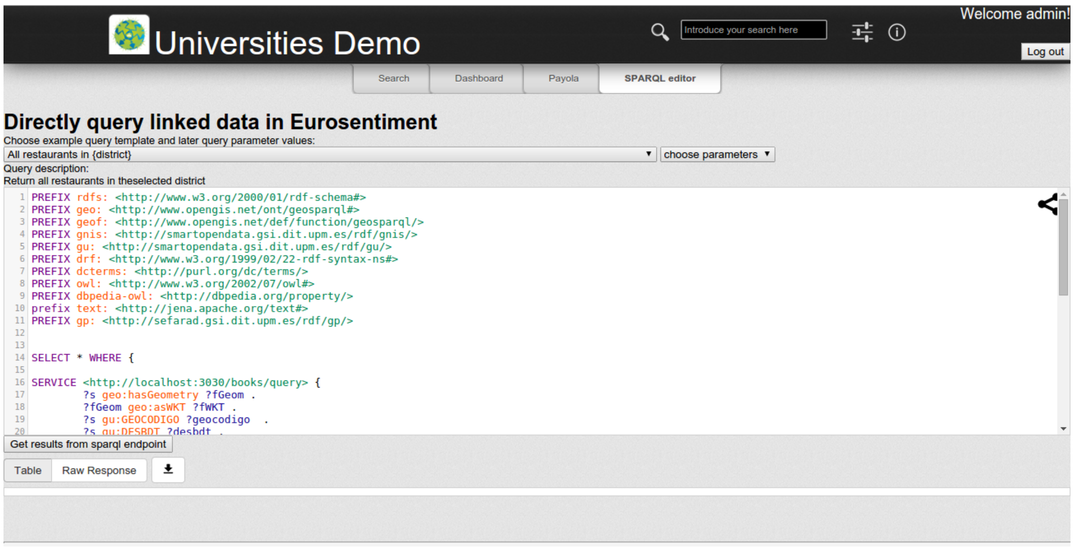
This sub-module, developed by Alejandro Saura Villanueva[1] allows the user to write an edit his own query. Those basic users with no full technical knowledge about SPARQL query language can make use of the SPARQL Editor based on *Yasqe*[2] provided by Sefarad. They can access it in one of the principal tabs of the application. This service provides a simple syntax highlighted text area, bundled with features such as autocompletion, and the option to query SPARQL endpoints. So, a non-advanced user can edit his own query with the assistance of the application. The complete list of features is presented below:

- Query syntax highlighting and checking
- Accessing -all- endpoints (including CORS-disabled ones, or endpoints on your localhost)

[1]http://www.gsi.dit.upm.es/index.php?option=com_jresearch&view=member&task=show&id=140&Itemid=193
[2]http://yasqe.yasgui.org/

- Prefix autocompletion (using prefix.cc[3])
- Endpoint search and autocompletion (Using CKAN[4] and the Mondeca Endpoint Status Catalogue[5])
- Query permalinks
- Persistent application states between user sessions
- Query bookmarking
- SNORQL-type navigation
- Works offline as well
- Configurable requests (e.g. for adding *soft-limit=-1* in queries to a 4-store endpoint)

The following screenshot shows the SPARQL Editor in Sefarad.



Directly query linked data in Eurosentiment

Choose example query template and later query parameter values:

All restaurants in {district} | choose parameters

Query description:

Return all restaurants in theselected district

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX gnis: <http://smartopendata.gsi.dit.upm.es/rdf/gnis/>
PREFIX gu: <http://smartopendata.gsi.dit.upm.es/rdf/gu/>
PREFIX drf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbpedia-owl: <http://dbpedia.org/property/>
prefix text: <http://jena.apache.org/text#>
PREFIX gp: <http://sefarad.gsi.dit.upm.es/rdf/gp/>


SELECT * WHERE {

SERVICE <http://localhost:3030/books/query> {
        ?s geo:hasGeometry ?fGeom .
        ?fGeom geo:asWKT ?fWKT .
        ?s gu:GEOCODIGO ?geocodigo  .
        ?s gu:DESBDT ?desbdt .
```

Figure 4.2: SPARQL Editor

On the other hand, those advanced users with technical knowledge about SPARQL syntax can write their queries without the help of the editor directly in the SPARQL section of the configuration window.

[3]http://prefix.cc/

[4]http://datahub.io/

[5]http://labs.mondeca.com/sparqlEndpointsStatus/

4.3.2 SPARQL queries executor

This module is responsible for executing the query and store the information retrieved. The users can select the information source (local dataset: Fuseki or Geoserver; or any SPARQL endpoint) and run the query. After receiving the response from the server, this module will index the retrieved data and will work with it using the rest modules.

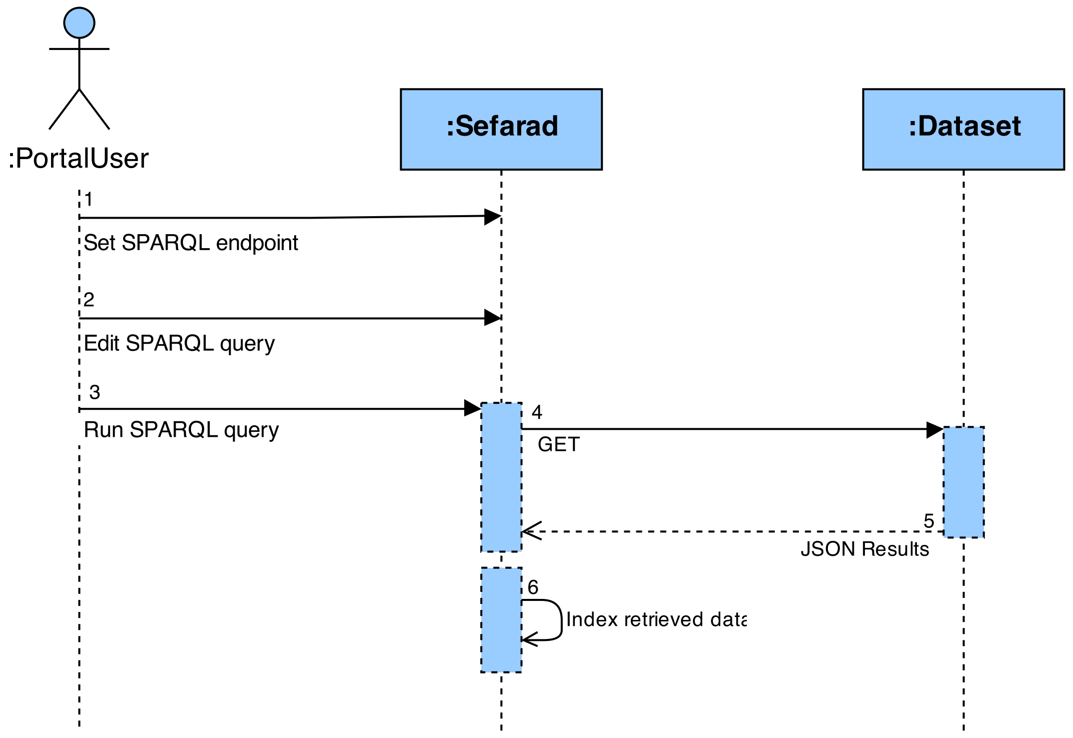


Figure 4.3: SPARQL query sequence diagram

4.4 Geo Proxy

The Geo Proxy Module is responsible for processing the data and converting it to a proper format suitable for its management with Sefarad. In the case of geographic information, it is particularly important the conversion from SPARQL response data to GeoJSON data, the proper format for representing geographic information in an *Openlayers map*. For this purpose, we have developed the next converter module shown in listing 4.1. The function takes a SPARQLJSON object and process it to return a GeoJSON object, which can be easily read and represented with an *Openlayers* map.

Listing 4.1: SPARQL to GeoJSON conversion

```
function sparqlToGeoJSON(sparqlJSON) {
  'use strict';
  var bindingindex, varindex, geometryType, wkt, coordinates,
      property;
  var geojson = {
    "type": "FeatureCollection",
    "features": []
  };

  for (bindingindex = 0; bindingindex < sparqlJSON.length; ++
      bindingindex) {

    for (var key in sparqlJSON[bindingindex]) {

      if ((sparqlJSON[bindingindex][key].datatype != undefined) &&
        (sparqlJSON[bindingindex][key].datatype() === "http://www.
          opengis.net/ont/geosparql#wktLiteral" ||
         sparqlJSON[bindingindex][key].datatype() === "http://www.
           openlinksw.com/schemas/virtrdf#Geometry")) {
        //assumes the well-known text is valid!
        wkt = sparqlJSON[bindingindehttp://tex.stackexchange.com/
          questions/180222/how-to-change-font-size-for-specific-
          lstlistingx][key].value();

        //find substring left of first "(" occurrence for geometry
          type
        switch (true) {
          case /POINT*/.test(wkt.substr(0, wkt.indexOf("("))):
            geometryType = "Point";
            coordinates = coordinates.substr(1, coordinates.length
                - 2); //remove redundant [ and ] at beginning and
                end
            break;
          case /MULTIPOINT*/.test(wkt.substr(0, wkt.indexOf("("))):
            geometryType = "MultiPoint";
            break;
          case /LINESTRING*/.test(wkt.substr(0, wkt.indexOf("("))):
            geometryType = "Linestring";
            break;
            case /MULTILINE*/.test(wkt.substr(0, wkt.indexOf("("))):
              geometryType = "MultiLine";
              break;
            case /POLYGON*/.test(wkt.substr(0, wkt.indexOf("("))):
              geometryType = "Polygon";
              break;
            case /MULTIPOLYGON*/.test(wkt.substr(0, wkt.indexOf("("))
              ):
              geometryType = "MultiPolygon";
              break;
            case /GEOMETRYCOLLECTION*/.test(wkt.substr(0, wkt.indexOf
              ("("))):
              geometryType = "GeometryCollection";
              break;
            default:
              //invalid wkt!
              continue;
          }

          var feature = {
            "type": "Feature",
            "geometry": {
              "type": geometryType,
              "coordinates": eval('(' + coordinates + ')')
            },
            "properties": sparqlJSON[bindingindex]
          };

          geojson.features.push(feature);
        }
      }
    }
    return geojson;
}
```

Listing 4.1: SPARQL to GeoJSON conversion

4.5 Local Server

The main feature of this module is to provide a local database to store our datasets. The user can upload his data to a local database to query and manage it with Sefarad, configuring it in the SPARQL tab in the configuration section, instead of using an external endpoint. It can be RDF data, for whose storage we use *Fuseki* and *Virtuoso*, or shapefiles data, for which we use *GeoServer*. Figure 4.4 shows the different ways to access these databases and represent the results with Openlayers.

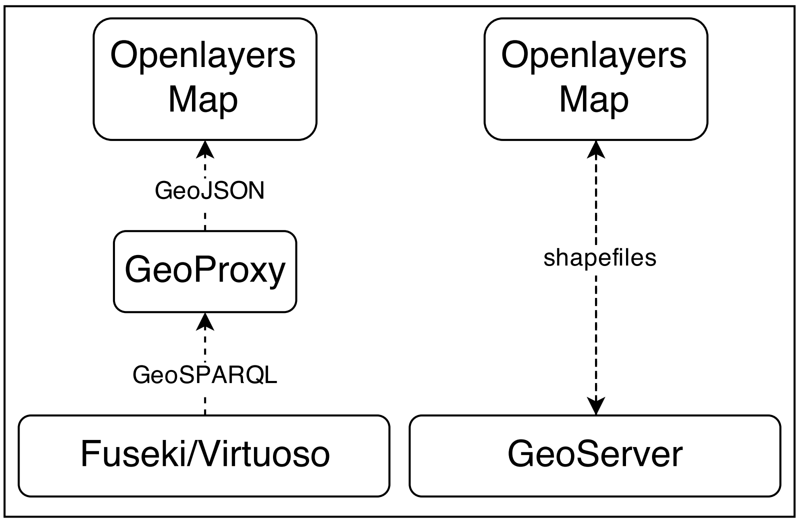


Figure 4.4: Fuseki/Virtuoso and GeoServer

4.5.1 Fuseki and Virtuoso

In either of these two databases we can store our datasets in RDF format. They both provide us an API REST to perform the queries and retrieve the data. Before storing the information in Sefarad, it should be treated in the *Geo Proxy*.

4.5.2 Geo Server

GeoServer[6] [6] is an open source software server written in Java that allows users to view and edit geospatial data.

Some of the main features of GeoServer are:

[6]http://geoserver.org/

- Full compatible with the OGC specifications.
- Easy installation and configuration (no large configuration files needed)
- Multiple formats supported, such as PostGIS or Shapefile.
- Multiple output formats supported, such as JPEG, GIF, PNG, SVG y GML.
- ECQL query language support.

GeoServer also includes an administration UI from which users can manage the stored data, observe and analyse the different attributes of the information as well as a preview of the different layers for what GeoServer includes an integrated OpenLayers client.

The user can save his geospatial dataset into a GeoServer installation and display the information with maps integrated into Sefarad as a new widget developed with OpenLayers.js. Into this server the user can upload geographic dataset in specific GIS formats such as DBF or Shapefile.

4.6 Search and filtering module

Once the information is retrieved and stored in a proper format, the 'Search and Filtering' module provides the necessary tools to categorize, filter and sort that information. For this purpose, this module provides different features:

- Back-end technology to manage and index the data, its different facets and the filtering features, updating the filtered results when the user selects new filtering criteria.
- A **results table** which shows the final filtered results and all the filters selected by the user, allowing him to manage both the filters and the results.
- **Keyword search box**, in which the user can introduce the words he wants to search by. This feature is fully detailed in subsection 4.6.1.
- 'Tag cloud' and 'vertical layout' **filtering widgets**, which show the different values for a particular facet, allowing the faceted search detailed in subsection 4.6.2. The user can configure all these widgets.
- Different many widgets to filter by numeric or geographic facets.

To manage and update the filtered data and the widgets in the layout when the user selects new filters, this module makes use of the advantages of *Knockout.js*. When a new filter is selected (either in a widget or in the search box), it is added to a collection that contains all the filters selected. When this collection changes, due to the knockout bindings, the *filtered data* is updated with the data that satisfy the new criteria. After that, all the active widgets are redrawn with the new filtered data available. Figure 4.5 and Figure 4.6 show examples of filtering.

4.6.1 Keyword search

This sub-module allows the user to filter the information by a keyword criteria. The user can introduce his search in a text-box shown in the top right corner of the application. This criterion will be used by the application to find results that contain the entered text in any of its fields.

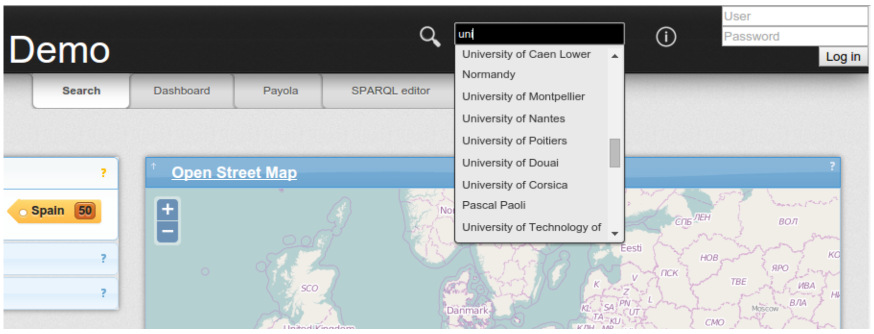
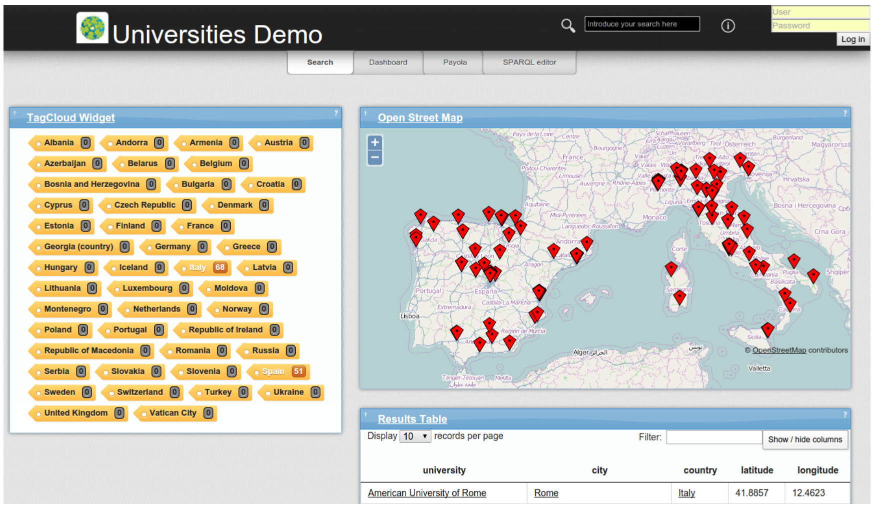


Figure 4.5: Keyword search example



university	city	country	latitude	longitude
American University of Rome	Rome	Italy	41.8857	12.4623

Figure 4.6: Faceted search example

4.6.2 Faceted search

This feature (also known as *faceted browsing*) allows the user to access the information according to a faceted classification system, applying multiple filters. A faceted classification system classifies each information element along multiple explicit dimensions, called facets, enabling the classifications to be accessed and ordered in multiple ways. Facets correspond to properties of the information elements. They are derived by analysis of the different keys of the retrieved data (in JSON format).

4.6.3 Geo filtering

Due to the geographical purpose of this master thesis, a geo filtering module is needed. This module works in a similar way of the two modules above, but focusing on geographical terms. Thus, this sub-module allows the management and filtering of the information on aspects such as latitude, longitude, area, etc.

With respect to the *Openlayers maps*, this module includes features such as layers classification, so that the different information can be categorized into different layers that can be shown or hidden on these maps (restaurants, hospitals, urban areas, etcetera.

Regarding to the data stored in GeoServer, this module includes a tool that allows the user to filter the information available on the server by using CQL or ECQL[7] query language. CQL (Common Query Language) is a query language created by the OGC for the Catalogue Web Services[8] specification. Unlike the XML-based Filter Encoding language, CQL is written using a familiar text-based syntax. It is thus more readable and better-suited for manual authoring. GeoServer provides an extended version of CQL called ECQL, which removes the limitations of CQL, providing a more flexible language with stronger similarities with SQL. The user can include in his query any of the attributes included in the layer, both text and numeric fields. Also, the user may employ any of the comparison operators (=, <>, >, <, >=, <=), filter by range (BETWEEN X AND Y). Comparisons can be established between an attribute and a value (numberic or textual), or between two attributes of the layer (e.g., ATTRIBUTE1 ¿ ATTRIBUTE2). Furthermore, this query language includes geometric filters capabilities, such as intersections or crosses. The full list of geometric predicates is: *EQUALS, DISJOINT, INTERSECTS, TOUCHES, CROSSES, WITHIN, CONTAINS, OVERLAPS, RELATE, DWITHIN, BEYOND.*

[7]http://docs.geoserver.org/2.5.x/en/user/filter/index.html
[8]http://www.opengeospatial.org/standards/cat

4.7 Model View View-Model

As we detailed before, we use the Knockout.js framework, which is a JavaScript library that helps us to create rich, responsive display and editor user interfaces with a clean underlying data model. Any time we have sections of UI that update dynamically, KO helps us implement it more simply and maintainably.

The Model View ViewModel (MVVM[9]) is an architectural pattern used in software engineering. Largely based on the model–view–controller pattern (MVC), MVVM is a specific implementation targeted at UI development platforms which support event-driven programming. The main features of this module are: **dependency tracking**, automatically updates the right parts of our UI (widgets layout) whenever our data model changes; **declarative bindings**, a simple way to connect parts of our UI to our data model.

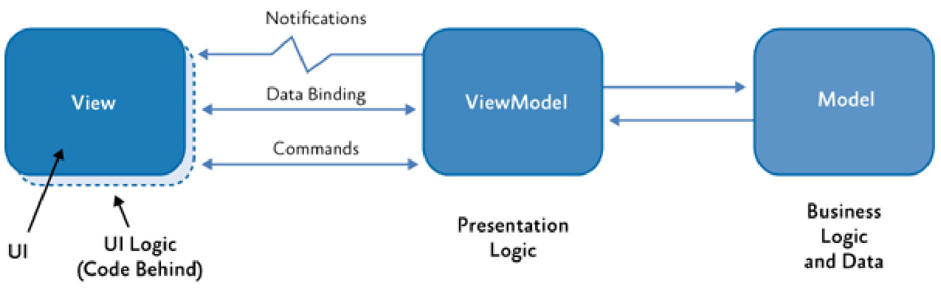


Figure 4.7: Model View ViewModel diagram[10]

MVVM facilitates a clear separation of the development of the graphical user interface (as markup language or GUI code) from the development of the back end logic known as the model (also known as the data model to distinguish it from the view model). The view model of MVVM is a *value converter*, meaning that the view model is responsible for exposing the data objects from the model in such a way that those objects are easily managed and consumed. In this respect, the view model is more model than view, and handles most if not all of the view's display logic (though the demarcation between what functions are handled by which layer is a subject of ongoing discussion and exploration). The view model may also implement a mediator pattern organising access to the backend logic around the set of use cases supported by the view.

MVVM was designed to make use of data binding functions in to better facilitate the separation of view layer development from the rest of the pattern by removing virtually all

[9]http://en.wikipedia.org/wiki/Model_View_ViewModel
[10]http://msdn.microsoft.com/

GUI code ("code-behind") from the view layer. Instead of requiring user experience (UX) developers to write GUI code, they can use the framework markup language and create bindings to the view model, which is written and maintained by application developers. This separation of roles allows interactive designers to focus on UX needs rather than programming of business logic, allowing for the layers of an application to be developed in multiple work streams for higher productivity.

By using knockout bindings in our HTML document, this module automatically updates the DOM elements whenever the `vm.filteredData()` array changes. An example of a knockout binding in Sefarad is show in listing[4.2].

```
<div id='column1tab0' class='column' >
  <!-- Widgets at right column -->
  <div class='container right' data-bind='template: { name: '
      widgets-template', foreach: activeWidgetsRight,  beforeRemove:
       function(elem) { $(elem).slideUp(1500,'easeOutBounce',
      function(){$(elem).remove(); }); }, templateOptions: {
      parentList: activeWidgetsRight} }, sortableList:
      activeWidgetsRight'></div>
</div>
```

Listing 4.2: Active widgets binding

The code above shows the data-binding for all the widgets at right column. As you can see at the `foreach: activeWidgetsRight` bindgin, the `widgets-template` is filled with every active widget. When the user adds a new widget, a new object is pushed. Due to de data-binding, the DOM object is automatically updated and a new HTML DIV for the new widget is added. That is how the model view view-model works.

4.7.1 Data Model

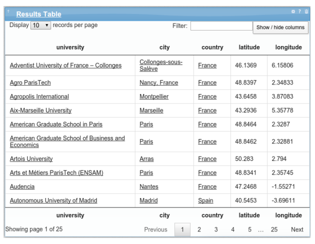
The view model is defined in the file `js/mvvm.js`. It is a "model of the view", meaning it is an abstraction of the view that also serves in mediating between the view and the model which is the target of the view data bindings. It could be seen as a specialized aspect of what would be a controller (in the MVC pattern) that acts as a converter that changes model information into view information and passes commands from the view into the model. The view model exposes public properties, commands, and abstractions.

So the view model contains of the public properties (functions and ko.observables) that will be accessed from the other parts of the project, directly or through data bindings.

4.7.2 Widgets layout

One of the main purposes of this project is to show the queried semantic data. To this end Sefarad includes a large library of widgets to display many kinds of information: filtering widgets, slider widget, graphic widgets (bars, wheels, donuts, etc.). To visualize geographic information, the most important widgets are: results table, filtering widgets (tagcloud and selector), Openlayers map for GeoJSON and Openlayers map for GeoServer shapefiles. We describe their main features below.

Results table. This widget shows the total filtered results. The user can show or hide the different columns of the data, sort the elements by any of the attributes and search any result. If any result contains a link to the URI of its web resource, it can be directly accessed by clicking on it, so it will open a new tab.

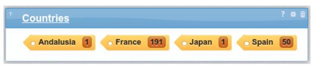

Results Table

Display 10 records per page Filter: Show / hide columns

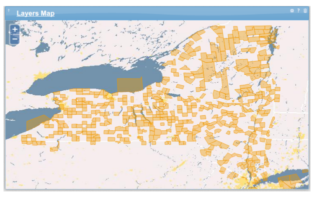
university	city	country	latitude	longitude
Adventist University of France – Collonges	Collonges-sous-Salève	France	46.1369	6.15806
Agro ParisTech	Nancy, France	France	48.8397	2.34833
Agropolis International	Montpellier	France	43.6458	3.87083
Aix-Marseille University	Marseille	France	43.2936	5.35778
American Graduate School in Paris	Paris	France	48.8464	2.3287
American Graduate School of Business and Economics	Paris	France	48.8462	2.32881
Artois University	Arras	France	50.283	2.794
Arts et Métiers ParisTech (ENSAM)	Paris	France	48.8341	2.35745
Audencia	Nantes	France	47.2468	-1.55271
Autonomous University of Madrid	Madrid	Spain	40.5453	-3.69611
university	city	country	latitude	longitude

Showing page 1 of 25 Previous 1 2 3 4 5 … 25 Next

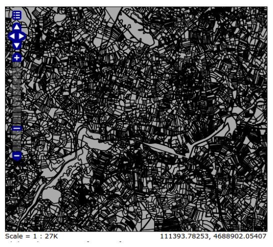
Tagcloud widget. This widget shows the total different values for the selected facet. It allows the user to select one o more values and filter the results, what we presented as *faceted filtering* or *faceted browsing*.



Map for GeoJSON data. This is an Openlayers map which can represent any geospatial and geometric information in GeoJSON format. It includes *POINT, MULTIPOINT, POLYGON, MULTIPOLYGON, LINE, etc.*. For this purpose, this widget uses the `geojson_format.read()` function provided by Openlayers.js. This map also includes the possibility of group the different types of data into different layers, so the user can show or hide the different kinds of results.



Map for GeoServer data. This map allows the user to display the different layers (file formats .dbf and .shp) stored on GeoServer. A important feature is that this widget includes a configuration section to manage ECQL queries in which the user can edit and run his own queries and display the results into the map.



Furthermore, Sefarad offers two different layout templates that the user can select according to his needs: a linear layout and an accordion layout.

4.7.2.1 Linear layout

The *linear layout* is the basic style layout for Sefarad. In this template the application is divided in two columns, with the right column width bigger than the left one. Each widget consists on its own division and item in the screen. If the *sortable widgets* option is enabled, any of the widgets can be placed either in both columns. As the right column is bigger, it is the best option to place the more visual widgets such as the different maps or the results table. In the other hand, the left column is better option for simpler widgets such as filtering widgets, which not require a large field of view.

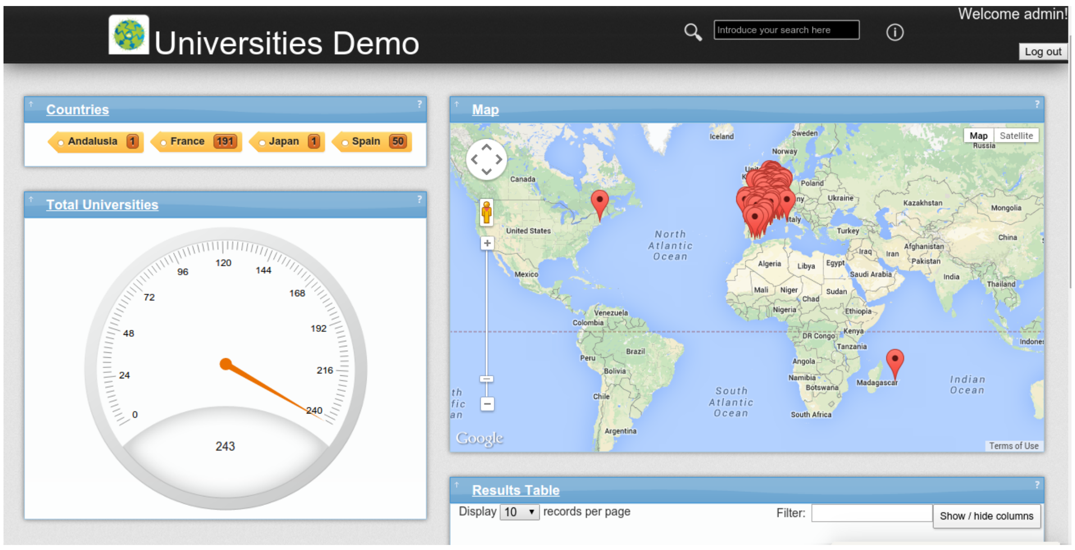


Figure 4.8: Linear Layout

4.7.2.2 Accordion layout

The *accordion layout* is a special style layout for Sefarad. It is also divided into two columns, each one with the same dimensions as in the previous case. The difference is that all the widgets in the left column are packed into an accordion widget. This option is specially designed to simplify the layout when the user wants to display a large amount of filtering widgets, so that the user can compress all these widgets into an accordion on the left of the screen, while the right column shows the results and map widgets.

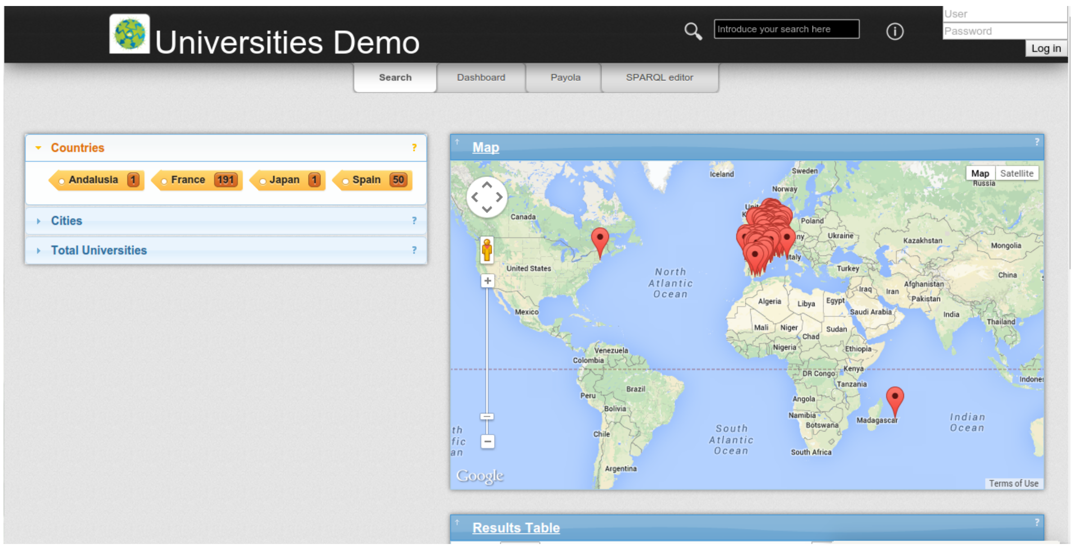


Figure 4.9: Accordion Layout

4.8 User management module

This module provides all the necessary features for users control and security. It includes an installation of a MongoDB database in which the application stores users authentication and authorization information (username, password, role and permissions) and users preferences and settings. This module also includes all the server-side technology necessary to manage user login/logout, load saved settings and saving new preferences. For this purpose, the system uses the PHP driver[11] for MongoDB databases management.

4.8.1 Security: authentication and authorization

As we introduced above, this module contains PHP sub-modules and a MongoDB collection in the Sefarad MongoDB database. For security reasons, the password of users are stored in the database md5 hashed.

For session handling, we have developed a Session Manager as explained in [10]. The security module provides a session manager in PHP, a module that will handle the HTTP session of a user in Sefarad, using MongoDB for storing the session data. It provides us important functionalities such as signing in a user (authentication), authorizing his/her actions, and logging him/her out. We use object-oriented programming principles for building

[11]http://php.net/manual/es/book.mongo.php

the module, so that it is easy to maintain. We also build a separate module for user authentication, which is used by the session manager for logging a user in.

We need to implement three php classes:

- **dbconnection.php**, detailed in section[4.8.1.1].
- **session.php**, detailed in section[4.8.1.2].
- **user.php**, detailed in section[4.8.1.3].

4.8.1.1 MondoDB connection

The class DBConnection handles the connection with the MongoDB database. You can see the code in the listing[4.3] below.

Listing 4.3: dbconnection.php

```php
<?php
class DBConnection {
  const HOST = 'localhost';
  const PORT = 27017;
  const DBNAME = 'sefarad';
  private static $instance;
  public $connection;
  public $database;

  private function __construct()  {
    $connectionString = sprintf('mongodb://%s:%d', DBConnection::
        HOST, DBConnection::PORT);
    try {
      $this->connection = new Mongo($connectionString);
      $this->database = $this->connection->selectDB(DBConnection::
          DBNAME);
    }
    catch(MongoConnectionException $e) {
      throw $e;
    }
  }

  static public function instantiate()  {
    if (!isset(self::$instance)) {
      $class = __CLASS__;
      self::$instance = new $class;
    }
    return self::$instance;
  }

  public function getCollection($name) {
    return $this->database->selectCollection($name);
  }
}
</div>
```

Listing 4.3: dbconnection.php

Calling the initialize() static method on this class returns an instance of it, and we can then select a collection by invoking the getCollection() method on this instance. The DBConnection class implements the Singleton[12] design pattern. This design pattern ensures that there is only a single connection open to MongoDB, within the context of a single HTTP request.

4.8.1.2 Session Manager

This module uses a collection in a MongoDB database for storing/retrieving/handling sessions.It extends the session handling with `session_set_save_handler()`, a function which allows us to define our own functions for storing and retrieving session data. The function takes six arguments, each one being the name of a callback function for handling sessions (open, close, read, write, destroy and gc). Let's see each callback in detail:

- **open()**: This method is called whenever a session is initiated with `session_start()`. It takes two arguments, the path to where the session will be stored and the name of the session cookie. It returns TRUE to indicate successful initiation of a session.
- **close()**: This is called at the successful end of a PHP script using session handling. This also needs to return TRUE.
- **read()**: This method is called whenever we are trying to retrieve a variable from the `$_SESSION` super global array. It takes the session ID as an argument and returns a string value of the `$_SESSION` variable.

[12]http://en.wikipedia.org/wiki/Singleton_pattern

- **write()**: This function is executed whenever we are trying to add or change something in `$_SESSION`. This takes the session ID and the serialized representation of the data to be stored in `$_SESSION` as its two arguments.
- **destroy()**: This is called whenever we are trying to terminate a session by calling the built-in `session_destroy()` method. It takes the session ID as its only parameter and returns TRUE upon success.
- **gc()**: This function is executed by the PHP session garbage collector. It takes the maximum lifetime of session cookies as its argument, and removes any session older than the specified lifetime. It also returns TRUE on success. The `session.gc_probability` setting in php.ini specifies the probability of the session garbage collector running.

The final code for the `session.php` class is shown in listing[4.4].

```
<?php
require_once ('dbconnection.php');

class SessionManager{

  consts declaration;
  private $_mongo, $_collection , $_currentSession;

  public function __construct() {
    $this->_mongo = DBConnection::instantiate();
    $this->_collection = $this->_mongo->getCollection(
        SessionManager::COLLECTION);
    session_set_save_handler(array(&$this,'open') , array(&$this,'
        close') , array(&$this,'read') , array(&$this,'write') ,
        array(&$this,'destroy') , array(&$this,'gc'));
    ini_set('session.gc_maxlifetime', SessionManager::
        SESSION_LIFESPAN);
    session_set_cookie_params(SessionManager::SESSION_LIFESPAN,
        SessionManager::SESSION_COOKIE_PATH, SessionManager::
        SESSION_COOKIE_DOMAIN);
    session_name(SessionManager::SESSION_NAME);
    session_cache_limiter('nocache');
    session_start();
  }

  public function open($path, $name) { return true; }
public function close() { return true; }

public function read($sessionId) {
  $query = array(
    'session_id' => $sessionId,
    'timedout_at' => array('$gte' => time()) ,
    'expired_at' => array('$gte' => SessionManager::
        SESSION_LIFESPAN));
  $result = $this->_collection->findOne($query);
  $this->_currentSession = $result;
  if (!isset($result['data'])) {return ''; }
  return $result['data'];
}

public function write($sessionId, $data) {
  $expired_at = time() + self::SESSION_TIMEOUT;
  $new_obj = array(
    'data' => $data,
    'timedout_at' => time() + self::SESSION_TIMEOUT,
    'expired_at' => (empty($this->_currentSession)) ? time() +
        SessionManager::SESSION_LIFESPAN : $this->_currentSession[
        'expired_at'] );
  $query = array('session_id' => $sessionId);
  $this->_collection->update($query, array(
    '$set' => $new_obj
  ) , array(
    'upsert' => True
  ));
  return True;
}

public function destroy($sessionId) {
  $this->_collection->remove(array('session_id' => $sessionId));
  return True;
}

public function gc() {
  $query = array('expired_at' => array('$lt' => time() ));
  $this->_collection->remove($query);
  return True;
  }

  public function __destruct() {
    session_write_close();
  }
}
$session = new SessionManager();
```

Listing 4.4: session.php

4.8.1.3 User class

Finally, this class represents a user in the web application. This class can be used to log a user in (authentication), enable him to view pages that he is allowed to see (authorization), and log him out when he wishes. The php class is detailed in Listing [4.5].

```
<?php
require_once ('dbconnection.php');
require_once ('session.php');

class User {
  const COLLECTION = 'users';
  private $_mongo;
  private $_collection;
  private $_user;

  public function __construct() {
    $this->_mongo = DBConnection::instantiate();
    $this->_collection = $this->_mongo->getCollection(User::
        COLLECTION);
    if ($this->isLoggedIn()) $this->_loadData();
  }

  public function isLoggedIn() {
    return isset($_SESSION['user_id']);
  }

  public function authenticate($username, $password) {
    $query = array('username' => $username,'password' => md5(
        $password));
```

```
      $this->_user = $this->_collection->findOne($query);
      if (empty($this->_user)) return False;
      $_SESSION['user_id'] = (string)$this->_user['_id'];
      return True;
    }

    public function logout() {
      unset($_SESSION['user_id']);
      unset($_SESSION['user_name']);
    }

    public function __get($attr) {
      if (empty($this->_user)) return Null;
      switch ($attr) {
      case 'username':
        $username = $this->_user['username'];
        return sprintf('Username: %s', $username);
      default:
        return (isset($this->_user[$attr])) ? $this->_user[$attr] :
            NULL;
      }
    }

    private function _loadData() {
      $id = new MongoId($_SESSION['user_id']);
      $this->_user = $this->_collection->findOne(array('_id' => $id))
          ;
    }
}
```

Listing 4.5: user.php

In the constructor of this class, we obtain a database connection and select the appropriate collection. These objects are stored in private member variables of the class. The `authenticate()` method of the class is used to authenticate a valid user. The method receives the username and password as its arguments. It queries the database with the username and MD5 hash of the password. If a matching document is found, the `ObjectId` of the document is casted to string and stored in `$_SESSION` as user_id. The method returns TRUE to indicate that the user is successfully authenticated. Otherwise the method returns FALSE.

The `isLoggedIn()` method checks whether the user is already logged in by simply checking the existence of user_id in `$_SESSION`. The `logout()` method terminates the authenticated session by unsetting the user_id field. If the user is logged in, the `loadData()` private method is called within the constructor to query the database with the ID and populate the values of user attributes. Finally, the `get()` method is used to read the attributes of a User object.

4.8.2 MongoDB: settings and preferences

This sub-module, as the one above, is composed of two parts: a MongoDB database for storing users' settings and preferences; and the PHP server-side technology to access the data. The MongoDB collection for this purpose (named *configurations*) can store as many settings documents as users registered in the system, but only on document per user.

That module contains three main php classes:

- **mongo_load.php**, detailed in section[4.8.2.1].
- **mongo_save.php**, detailed in section[4.8.2.2].
- **mongo_delete.php**, detailed in section[4.8.2.3].

4.8.2.1 Loading configuration

```php
<?php
  // connect
  $m = new MongoClient();
  // select Sefarad Database
  $db = $m->sefarad;
  // select Configuration collection
  $collection = $db->configuration;
  // search saved configuration
  $query = array( 'name' => 'saved_configuration',
          'user_id' => $_SESSION['user_id']);
  $cursor = $collection->find( $query );

  // load configuration (saved or default)
  if (($cursor->count()) > 0) {
    foreach ($cursor as $doc) {
        echo (json_encode(($doc)));
    }
  } else {
    $query = array( 'name' => 'default_configuration' );
    $cursor = $collection->find( $query );

    if (($cursor->count()) > 0) {
      foreach ($cursor as $doc) {
          echo (json_encode(($doc)));
      }
    } else {
      trigger_error("No configuration found", E_USER_ERROR);
    }
  }
?>
```

Listing 4.6: mongo_load.php

First, the script establishes connection to 'sefarad' database and then, it selects the 'configuration' collection. After the connection, we query the collection is queried searching a saved configuration for the current logged in user. That query includes the user id. If any saved configuration is found, it is returned in JSON format. Otherwise, the script returns the default configuration to initialize Sefarad.

4.8.2.2 Saving configuration

```php
<?php

require ('../auth/session.php');

if (isset($_SESSION['user_id'])){

  $ac = $_REQUEST['actual_configuration'];

  // connect to Mongo
  $m = new MongoClient();
  // select Sefarad DataBase
  $db = $m->sefarad;
  // select Configuration collection
  $collection = $db->configuration;

  // delete old saved configuration
  $collection->remove(array(
    'name' => 'saved_configuration',
    'user_id' => $_SESSION['user_id'])
  );

  // save new configuration
  $document = json_decode($ac,true);

  unset($document['_id']);

  $document['user_id'] = $_SESSION['user_id'];

  $collection->insert($document);
}

?>
```

Listing 4.7: mongo_save.php

In the case of saving configuration, after connecting to the database in the same way as in the case above, the script deletes the old saved configuration for the current user (if any). After that, the script inserts the new configuration into a new document into the configuration collection.

4.8.2.3 Deleting configuration

```php
<?php

require ('../auth/session.php');

if (isset($_SESSION['user_id'])){

  // connect to Mongo
  $m = new MongoClient();

  // select Sefarad DataBase
  $db = $m->sefarad;

  // select Configuration collection
  $collection = $db->configuration;

  // delete old saved configuration
  $collection->remove(array(
    'name' => 'saved_configuration',
    'user_id' => $_SESSION['user_id'])
  );
}

?>
```

Listing 4.8: mongo_delete.php

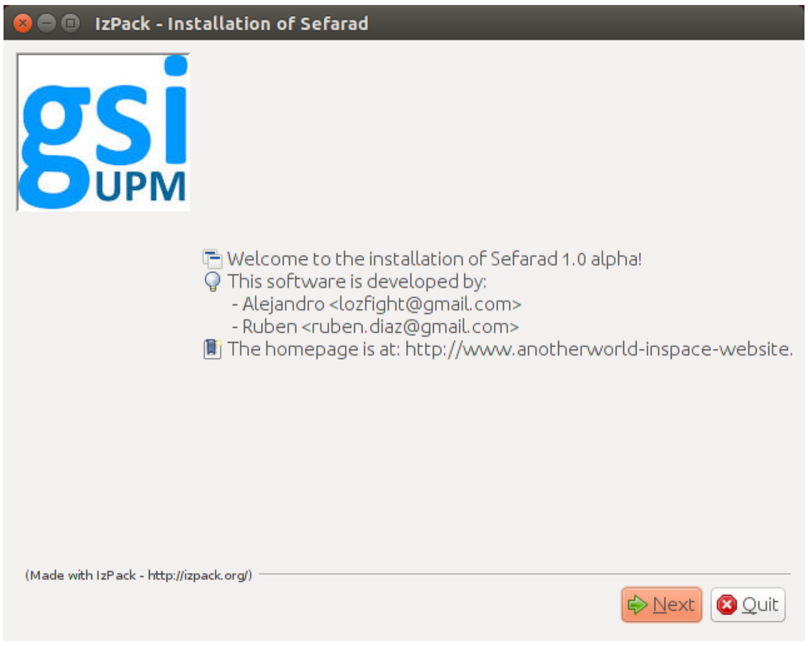
Finally, we develop the `mongo_delete.php` class. This script will be called when the user wants to reset his configuration. The main purpose of this script is deleting the saved configuration for the current user, so it simply connects to MongoDB and deletes this document by querying it with the user id.

4.9 Setup module

Since the web application consists of several modules, some of which are external, we have facilitated the installation providing an `Setup.jar` installer. It allows you to customize your installation by selecting which modules you want to install, to select where to copy all the needed files and it guide you during the installation.

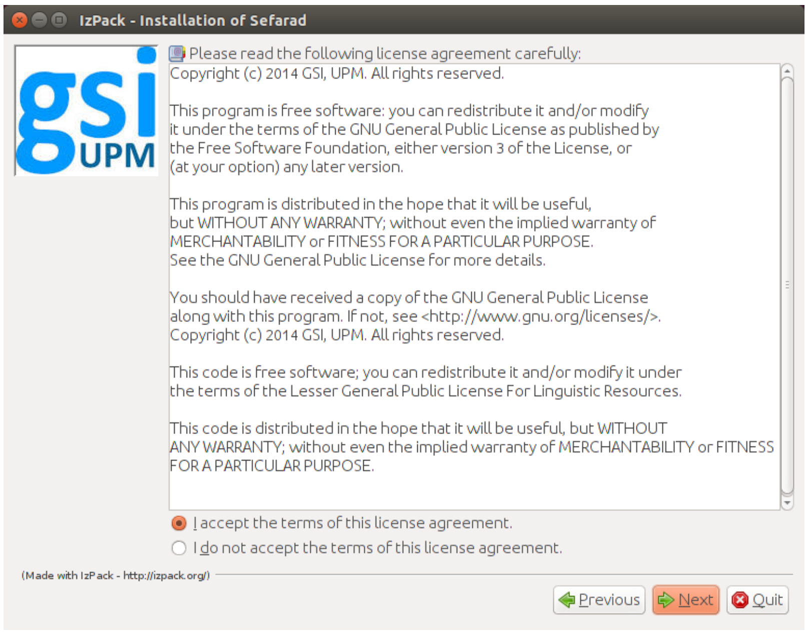
4.9.1 Custom installer

To facilitates the installation in any computer with a Linux OS installed, Sefarad includes an installer. By executing *Setut.jar* file, you will install a complete version of the application in your computer. Next images show some of the steps during the installation.


IzPack - Installation of Sefarad
gsi UPM
(Made with IzPack - http://izpack.org/)
Next
Quit

Welcome to the installation of Sefarad 1.0 alpha!
This software is developed by:
- Alejandro <lozfight@gmail.com>
- Ruben <ruben.diaz@gmail.com>
The homepage is at: http://www.anotherworld-inspace-website.

Figure 4.10: Installation example (1)


IzPack - Installation of Sefarad
gsi UPM
(Made with IzPack - http://izpack.org/)
Previous
Next
Quit

Please read the following license agreement carefully:
Copyright (c) 2014 GSI, UPM. All rights reserved.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.
Copyright (c) 2014 GSI, UPM. All rights reserved.

This code is free software; you can redistribute it and/or modify it under
the terms of the Lesser General Public License For Linguistic Resources.

This code is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE.

[✓ I accept the terms of this license agreement.]
I do not accept the terms of this license agreement.

Figure 4.11: Installation example (2)

We provide also a custom installer that allows the user to select what widgets he wants to install, in order to avoid installing elements that will not be used by the user.

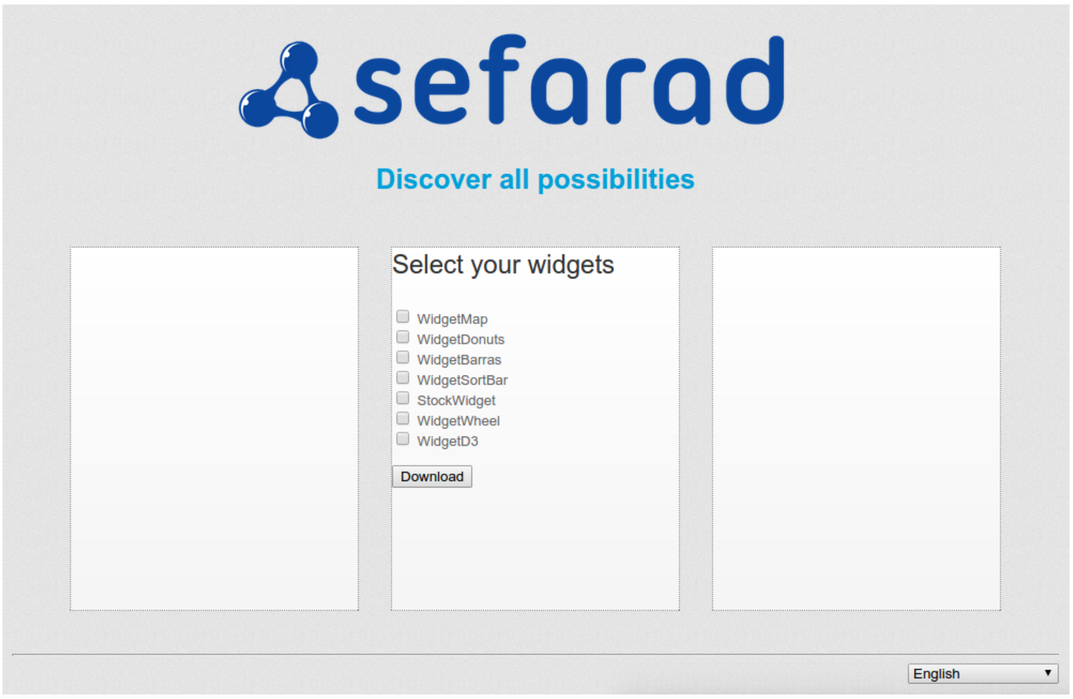


Figure 4.12: Custom installer

For this module, we have developed a PHP script that iterates through every widget contained in `widgets/` folder and showing them for selection. After the user have selected the widgets the deserved to include in his own installation, the script includes them and all the base files into a .zip file and download it into the user's computer.

4.9.2 Automation: Grunt.JS Task Runner

Since the web application consists of several modules, we need to maintain a logical directory, files and code structure in order to ensure our project base is manageable and maintainable. The final project consists of all these modules, directories and files, which also are related to each others. To build this final built project, we need to perform repetitive task. With *Grunt.js task runner* we can automate all this tasks invoking a simple command.To automate with Grunt, first we need to declare the needed dependencies in a `package.json`[4.9] file.

```json
{
  "name": "Sefarad",
  "version": "0.0.1",
  "description": "Linked Data Visualization Framework",
  "main": "Gruntfile.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "devDependencies": {
    "grunt": "~0.4.2",
    "grunt-contrib-concat": "~0.3.0",
    "grunt-processhtml": "~0.3.0",
    "grunt-contrib-copy": "~0.5.0",
    "grunt-contrib-clean": "~0.5.0"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/gsi-upm/demo-smartopendata.git"
  },
  "keywords": [
    "Sefarad",
    "LinkedData",
    "Visualization",
    "Framework"
  ],
  "author": "GSI-UPM",
  "bugs": {
    "url": "https://github.com/gsi-upm/Sefarad/issues"
  }
}
```

Listing 4.9: package.json

As you can see in the code above, we have included all the needed dependencies in the key `devDependencies`, including Grunt.js. This file also contains several information about the application.

The file containing all the automated tasks is `Gruntfile.js`[4.10] shown below.

Listing 4.10: Gruntfile.js

```
// Do grunt-related things in here
module.exports = function(grunt) {

  // Project configuration.
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    processhtml: {
      templates: {
        files: {
          'build/index.php': ['src/sefarad.php']
        }
      },
      php_widgets: {
        files: {
          'build/sefarad.php': ['src/sefarad.php']
        }
      },
      universitiesDemo: {
        files: {
          'build/js/mvvm.js': ['src/js/mvvm.js']
        }
      },
    },
    copy: {
      main: {
        expand: true,
        cwd: 'src/',
        src: ['ajax-solr/**','css/**','img/**','js/**','
            json_examples/**', 'php/**','auth/**','sefarad.php','!js
            /widgets/widget_template.js'],
        dest: 'build/',
      },
      universitiesDemo: {
        expand: true,
        cwd: 'src/demos/universitiesDemo/',
        src: 'demo.html',
        dest: 'build/',
      },
    },
    clean: {
      build: {
        src: ['build/*','!.gitignore'],
      }
    },
  });

  // Load plugins and tasks.
  grunt.loadTasks('grunt_tasks');
  grunt.loadNpmTasks('grunt-contrib-concat');
  grunt.loadNpmTasks('grunt-processhtml');
  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-contrib-clean');

  // Tasks.
  grunt.registerTask('default', ['clean:build','processhtml:
      templates','include-all-widgets','copy:main']);
};
```

Listing 4.10: Gruntfile.js

Most Grunt tasks rely on configuration data defined in an object passed to the `grunt.initConfig()` method. In this file, `grunt.file.readJSON('package.json')` imports the JSON metadata stored in `package.json` into the grunt config. Each task expects its configuration to be specified in a property of the same name. Here, we have specified and configured three tasks: processhtml, copy and clean. Each one of these tasks requires its own configuration parameters in json format.

After that, we have to load the different plugins and tasks. Grunt available plugins are specified in `package.json`[4.9] as dependencies, so they have been installed via `npm install` and may be enabled with a simple command, such as `grunt.loadNpmTasks('grunt-processhtml')`. For enabling our custom defined tasks (i.e. `include_widgets.js`), we run `grunt.loadTasks('grunt_tasks')`, which will enable all the task files in `grut_tasks/` directory.

Finally, we define our custom task commands, specifying the tasks which will run when the user execute the command and the order of execution.

CHAPTER 5

Case Study

In this chapter we are going to describe a selected use cases by explaining the running of all the tools involved and its purpose and responses. Thank to these use cases, we will show the overall performance of the application and all the main functions available to the user.

5.1 Introduction

To show the different features and tools Sefarad provides, we have experienced three different case studies, each of which helps us to show different aspects of the application. These three case studies are:

- **European Universities.** In this study, we run a SPARQL query to the DBpedia endpoint querying about all universities in European countries. In this case study we:
  - Select a SPARQL endpoint.
  - Edit and run our own SPARQL query.
  - Display the retrieved data in a graphical table.
  - Point the geographical position of the results into an *Open Street Map*[1] by using *Openalayers.js* framework.
  - Add some filtering widgets to test faceted search and keyword search
  - Edit and run some SPARQL queries in the *dashboard* tab for managing some static values.
- **Restaurants and Districts in Madrid.** In this study, we run a SPARQL query to our local dataset deployed in a Fuseki installation. We query about restaurants in Madrid with some piece of information such as price, ranking, food type, etcetera. The result data will contain a RDF triple containing GeoJSON information about the city district in which it is located, allowing us to represent polygons in *Openlayers*. In this case study we:
  - Query a Fuseki dataset.
  - Display the accordion layout to include many facets for filtering.
  - Display GeoJSON polygons in an OpenLayers map.
- **Slovakian parcels.** In this case study we will work with the dataset provided by the Slovak Environmental Agency (SEA[2]) about harmonised protected sites dataset according to INSPIRE[3] Data Specification on Protected Sites – Guidelines through WFS service interface. In this case we will also work with GeoJSON data as in the case before, so we will test about the same features, but the purpose of this case is to test the performance of the application with a more professional and standardized dataset. In other words, to show a professional GIS application of Sefarad.

[1] http://www.openstreetmap.org/
[2] http://www.sazp.sk/
[3] http://inspire.ec.europa.eu/

5.2 European universities

In this case study we will query, index, classify, filter and display Linked Data relating to European Universities available in DBpedia. This is a very simple case of study that will help us to try the main features of Sefarad, including all those relative to query a SPARQL endpoint, manage the data with different filtering methods and display the final results in some graphical and geographical widgets. The following sub-sections will guide you through each of the steps with the help of listings, graphics and screenshots.

5.2.1 Query and retrieve the data: SPARQL

At this point, we select an external endpoint: *DBpedia*[4]. We run the following SPARQL query to retrieve all the universities in Europe with its name and location (country, city and geographic coordinates).

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX dbpedia: <http://dbpedia.org/>
PREFIX dbpedia2: <http://dbpedia.org/ontology/>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>

SELECT ?university ?country ?city ?latitude ?longitude

WHERE {
  ?universityResource rdf:type dbpedia2:University ;
  dbpedia2:country ?countryResource ;
  dbpedia2:city ?cityResource ;
  rdfs:label ?university ;
  geo:lat ?latitude ;   geo:long ?longitude .
  ?countryResource rdfs:label ?country ;
  dc:subject dbpedia:Countries_in_Europe.
  ?cityResource rdfs:label ?city

  FILTER ( lang(?university) = 'en' )
}
```

Listing 5.1: European universities SPARQL

[4] http://dbpedia.org/

An example piece of the information retrieved is shown below.

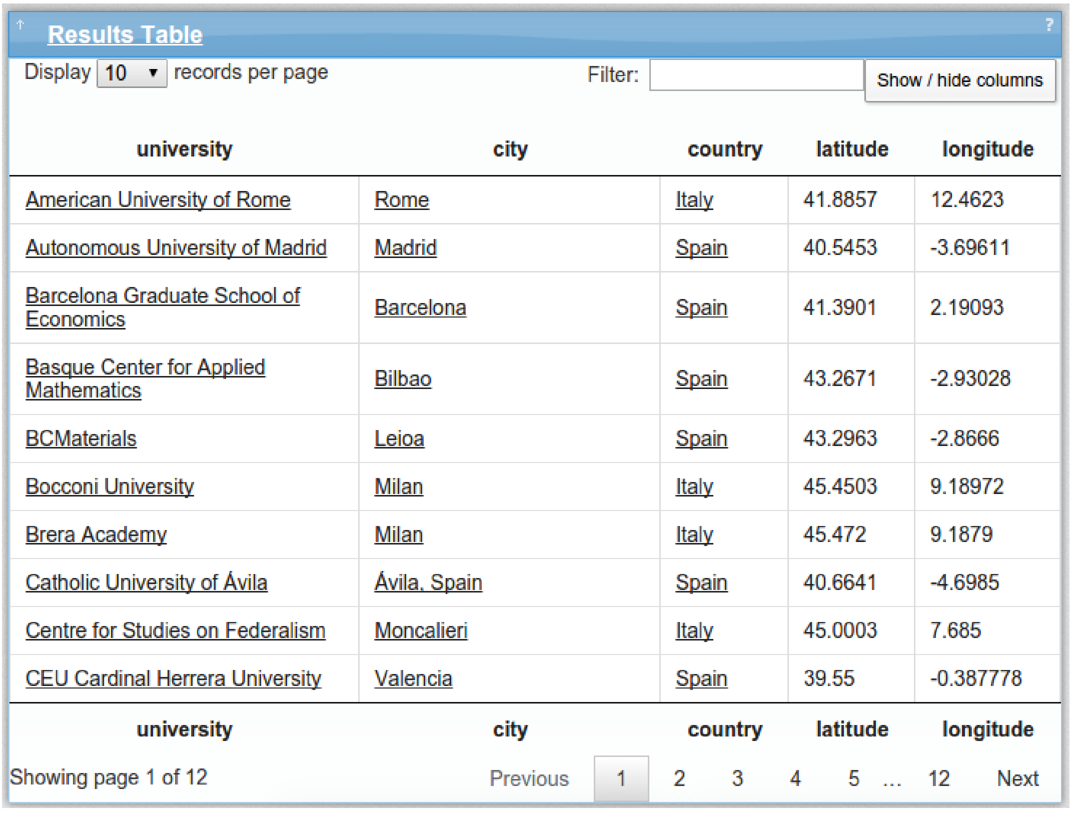
```
{
  "university": {
    "type": "literal",
    "xml:lang": "en",
    "value": "University of Barcelona"
  },
  "city": {
    "type": "literal",
    "xml:lang": "en",
    "value": "Barcelona"
  },
  "country": {
    "type": "literal",
    "xml:lang": "en",
    "value": "Spain"
  },
  "latitude": {
    "type": "typed-literal",
    "datatype": "http://www.w3.org/2001/XMLSchema#float",
    "value": "41.3867"
  },
  "longitude": {
    "type": "typed-literal",
    "datatype": "http://www.w3.org/2001/XMLSchema#float",
    "value": "2.16389"
  }
}
```

Listing 5.2: Universities results JSON example

We retrieve the response data in JSON format. As you could suppose, each one of the results contains five keys: university, country, city, latitude, longitude; as we indicate in the *SELECT* sentence of the query. Once the information is retrieved, the application stores and indexes it, obtaining the different facets for future faceted search work.

5.2.2 Showing the data: results table

The main widget available in Sefarad for exploring and ordering the final results is the *Results Table* widget. You can add it from the *'Add new widget'* section. Once you have done it, the widget automatically recognises the different facets of the results and draws them into a table like the one above (Figure[5.2]).


Results Table

Display 10 records per page Filter: Show / hide columns

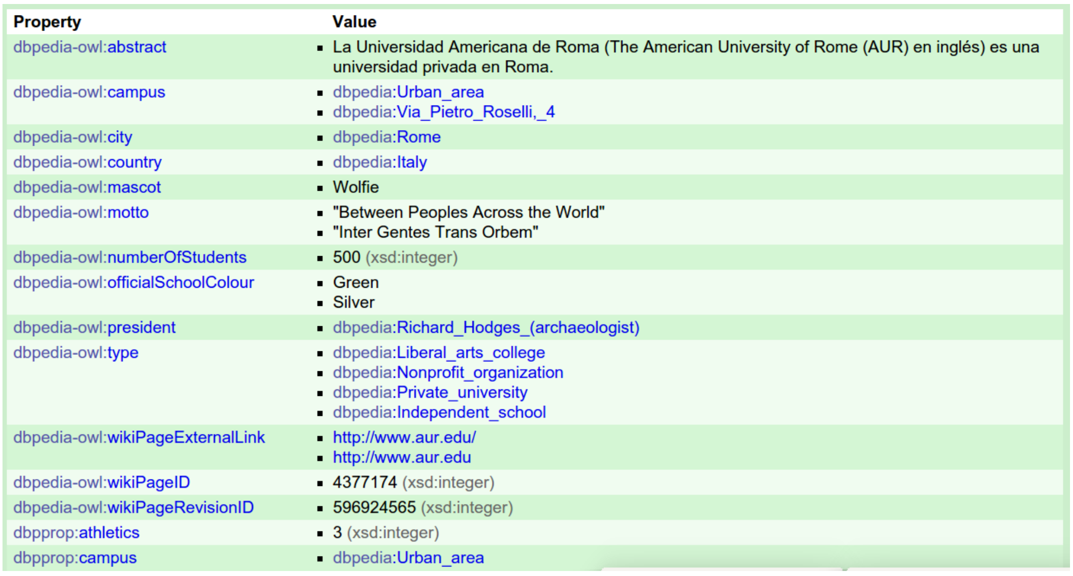
university	city	country	latitude	longitude
American University of Rome	Rome	Italy	41.8857	12.4623
Autonomous University of Madrid	Madrid	Spain	40.5453	-3.69611
Barcelona Graduate School of Economics	Barcelona	Spain	41.3901	2.19093
Basque Center for Applied Mathematics	Bilbao	Spain	43.2671	-2.93028
BCMaterials	Leioa	Spain	43.2963	-2.8666
Bocconi University	Milan	Italy	45.4503	9.18972
Brera Academy	Milan	Italy	45.472	9.1879
Catholic University of Ávila	Ávila, Spain	Spain	40.6641	-4.6985
Centre for Studies on Federalism	Moncalieri	Italy	45.0003	7.685
CEU Cardinal Herrera University	Valencia	Spain	39.55	-0.387778
university	city	country	latitude	longitude

Showing page 1 of 12 Previous 1 2 3 4 5 … 12 Next

Figure 5.1: Results Widget with universities

As we work with large amounts of data, we realized the need to do this widget as much configurable as possible, so that the user can select how many items to display per section, which columns to show or hide, etcetera. Furthermore, we included in the results table a search box that the user can use to search within the filtered results, which is an extra functionality apart from filtering technologies. All those configuration options can be shown in the figure above.

Moreover, if you have included the URI of the different resources in the SPARQL query, you can browse the results by clicking on them in the results table. For example, by clicking on *American University of Rome*, the following web page is opened.



Property	Value
dbpedia-owl:abstract	▪ La Universidad Americana de Roma (The American University of Rome (AUR) en inglés) es una universidad privada en Roma.
dbpedia-owl:campus	▪ dbpedia:Urban_area ▪ dbpedia:Via_Pietro_Roselli,_4
dbpedia-owl:city	▪ dbpedia:Rome
dbpedia-owl:country	▪ dbpedia:Italy
dbpedia-owl:mascot	▪ Wolfie
dbpedia-owl:motto	▪ "Between Peoples Across the World" ▪ "Inter Gentes Trans Orbem"
dbpedia-owl:numberOfStudents	▪ 500 (xsd:integer)
dbpedia-owl:officialSchoolColour	▪ Green ▪ Silver
dbpedia-owl:president	▪ dbpedia:Richard_Hodges_(archaeologist)
dbpedia-owl:type	▪ dbpedia:Liberal_arts_college ▪ dbpedia:Nonprofit_organization ▪ dbpedia:Private_university ▪ dbpedia:Independent_school
dbpedia-owl:wikiPageExternalLink	▪ http://www.aur.edu/ ▪ http://www.aur.edu
dbpedia-owl:wikiPageID	▪ 4377174 (xsd:integer)
dbpedia-owl:wikiPageRevisionID	▪ 596924565 (xsd:integer)
dbpprop:athletics	▪ 3 (xsd:integer)
dbpprop:campus	▪ dbpedia:Urban_area

Figure 5.2: American University of Rome DBpedia webpage

5.2.3 Geographic representation: Openlayers map

The main widget available in Sefarad for representing geospatial information is the *Openlayers map* widget. In this case, since we only have coordinates information (latitude and longitude) we can only represent the universities' position points in the map. In the image below, we have filtered the results by country (Spain and Italy) for a better appreciation.

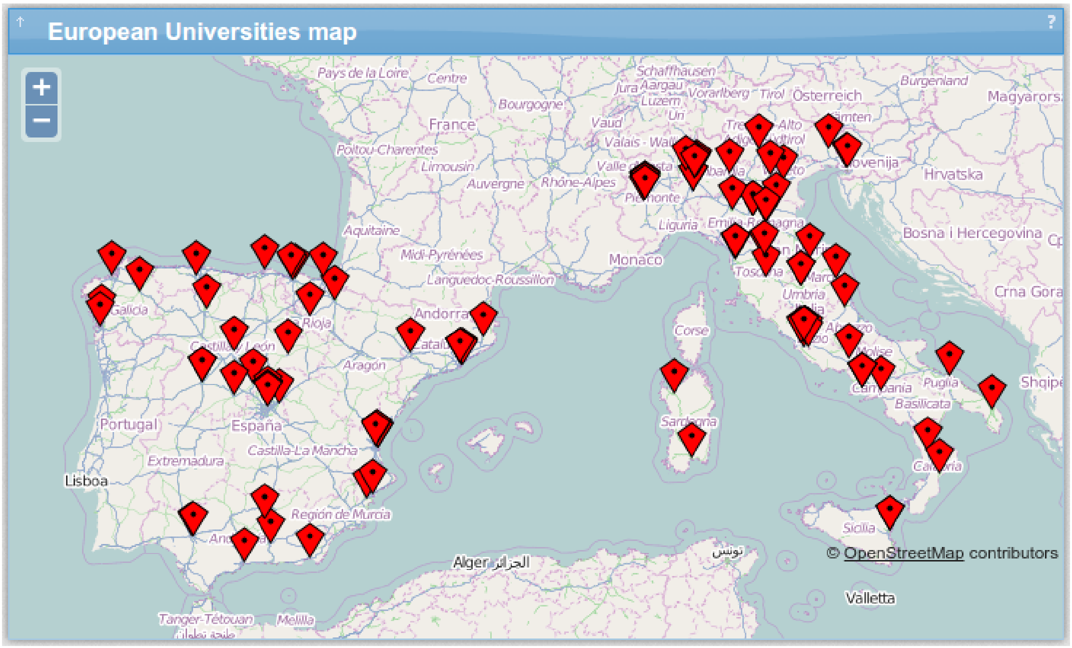


Figure 5.3: Universities in Spain and Italy

5.2.4 Filtering technologies

The last feature we will experience in this case study is both filtering technologies: faceted and keyword. To use keyword filtering, we must enter our keywords into the search box included at the right top of the application. As we write, the application will give us options that match what we have introduced, as shown in the image below.

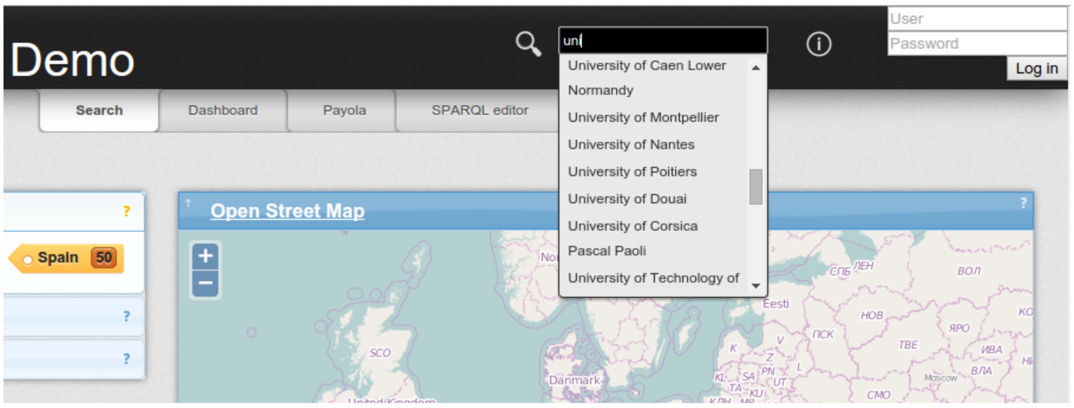
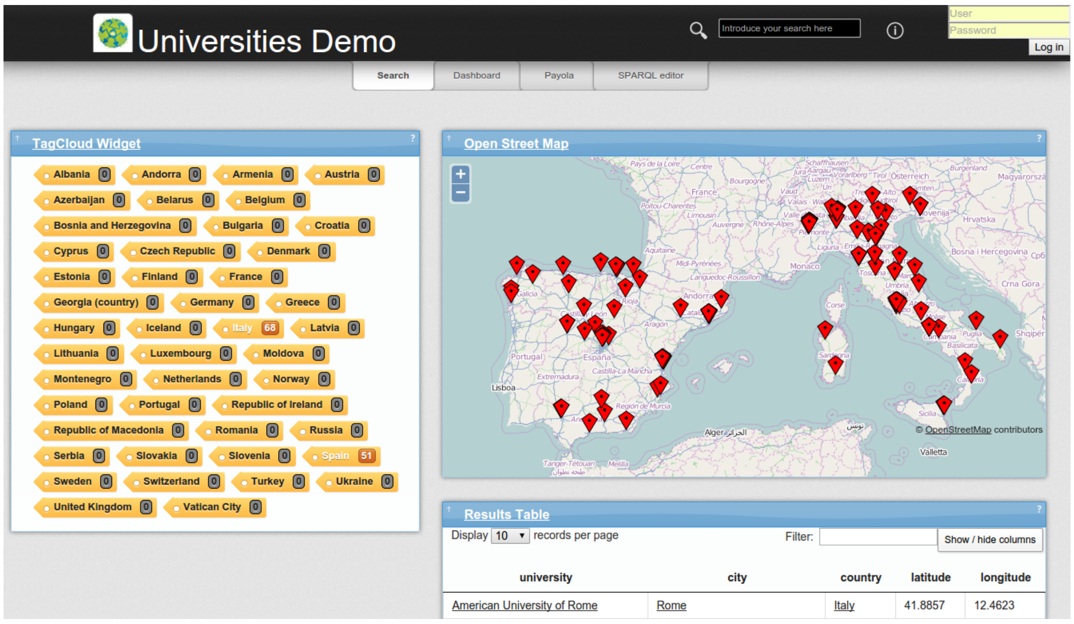


Figure 5.4: Keyword search example

To use faceted search, we must add some widgets that show the different possible values for the selected facet. After adding this widget for the *country* facet, the final layout display for this *Universities Demo* is shown in Figure [5.6].



university	city	country	latitude	longitude
American University of Rome	Rome	Italy	41.8857	12.4623

Figure 5.5: Universities Demo Layout

5.3 Restaurants and Districts in Madrid

In this study, we run a SPARQL query to our local dataset deployed in a Fuseki installation. We query about restaurants in Madrid[5][6] with some piece of information such as price, ranking, food type, etcetera. Result data contains a RDF triple containing GeoJSON information about the district in which it is located, allowing us to represent polygons in *Openlayers*.

5.3.1 Download and process the information

In this case, we download the data[7][8] and process it in order to upload it to our Fuseki database. The following graph shows this process in a clear way.

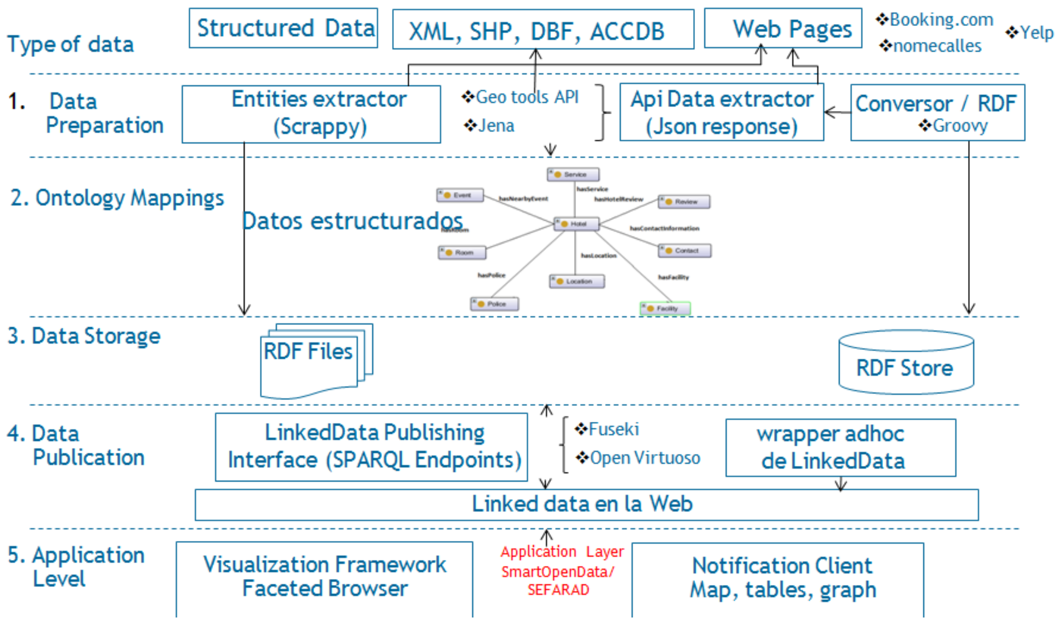


Figure 5.6: Processing restaurants data

[5]http://www.madrid.org/nomecalles/
[6]http://www.yelp.com/madrid
[7]http://www.madrid.org/nomecalles/
[8]http://www.yelp.com/madrid

5.3.2 SPARQL and Fuseki

In this case, we will not query a SPARQL endpoint of an online dataset. We will store the information in RDF format on a local server (Fuseki) and we will query it from Sefarad. We have created two new databases in Fuseki: *districts* and *restaurants*.

Listing 5.3: Restaurants results JSON example

```
"s": {
  "type": "uri",
  "value": "http://smartopendata.gsi.dit.upm.es/rdf/gu/Features
      /773372"
},
"fGeom": {
  "type": "uri",
  "value": "http://smartopendata.gsi.dit.upm.es/rdf/gu/Geometries
      /90478c001031d2ab9e9c199257ecbbb2724edb77"
},
"fWKT": {
  "datatype": "http://www.opengis.net/ont/sf#wktLiteral",
  "type": "typed-literal",
  "value": "MULTIPOLYGON (((444197.329 4477208.2759, ...  ,
      444197.329 4477208.2759)))"
},
"geocodigo": {
  "type": "literal",
  "value": "7904"
},
"desbdt": {
  "type": "literal",
  "value": "Salamanca"
},
"dbpediaLink": {
  "type": "uri",
  "value": "http://dbpedia.org/resource/Salamanca_(Madrid)"
},
"d": {
  "type": "uri",
  "value": "http://sefarad.gsi.dit.upm.es/rdf/gp/restaurants/casa-
      julian-madrid-2"
},
"p": {
  "type": "uri",
  "value": "http://sefarad.gsi.dit.upm.es/rdf/gp/district"
},
"o": {
  "type": "literal",
  "value": " Salamanca "
},
"price": {
  "type": "literal",
  "value": ""
},
"foodtype": {
  "type": "literal",
  "value": " Spanish "
},
"stars": {
  "type": "literal",
  "value": "4.5"
},
"reservations": {
  "type": "literal",
  "value": " Takes Reservations No "
},
"takeout": {
  "type": "literal",
  "value": "No"
},
"lat": {
  "type": "literal",
  "value": "40.429118500000001"
},
"long": {
  "type": "literal",
  "value": "-3.6858382999999999"
}
```

Listing 5.3: Restaurants results JSON example

As you can see in the image above, which shows the piece of information related to one restaurant within the retrieved data, each restaurant contains geographical information as a *MULTIPOLYGON* value, so we can draw it in a map.

The SPARQL query for this case study is shown below.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX gnis: <http://smartopendata.gsi.dit.upm.es/rdf/gnis/>
PREFIX gu: <http://smartopendata.gsi.dit.upm.es/rdf/gu/>
PREFIX drf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbpedia-owl: <http://dbpedia.org/property/>
prefix text: <http://jena.apache.org/text#>
PREFIX gp: <http://sefarad.gsi.dit.upm.es/rdf/gp/>

SELECT * WHERE {

  SERVICE <http://localhost:3030/districts/query> {
        ?s geo:hasGeometry ?fGeom .
        ?fGeom geo:asWKT ?fWKT .
        ?s gu:GEOCODIGO ?geocodigo  .
        ?s gu:DESBDT ?desbdt .
        ?s owl:sameAs ?dbpediaLink .
  }

  SERVICE <http://localhost:3030/restaurants/query> {
    ?d ?p ?o
    FILTER(REGEX(?o, ?desbdt))
  }

  SERVICE <http://localhost:3030/restaurants/query> {
    ?d gp:price ?price .
    ?d gp:foodtype ?foodtype .
    ?d gp:stars ?stars .
  }
}
```

Listing 5.4: European universities SPARQL

We have run a query that allows us to retrieve information from various data types (districts and restaurants). In addition, we have combined the information available in our dataset with information available in DBpedia, so what gives us access to more facets of the data.

5.3.3 Faceted search

As in the case of European universities, we will experience faceted search. However, in this case we have a greater number of facets, allowing us to observe the results when combining it. After adding a *Tag Cloud* widget for each of the facets we want to filter by, we have the next application layout.

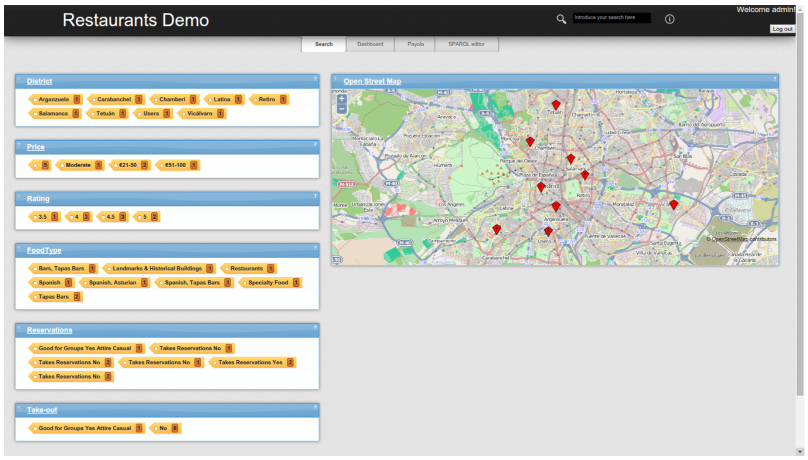


Figure 5.7: Linear layout

As you can see in the image above, when the number of facets becomes higher and higher, it becomes an impractical interface, because we can not show all the widgets in the window at the same time, and if we filter by a facet whose widget is lower, we must 'remove' the display map, so we can not see how the results are updated. To solve this problem, we developed another type of layout: the accordion layout [4.7.2.2]. Applying respective layout, the previous screen becomes as follows.

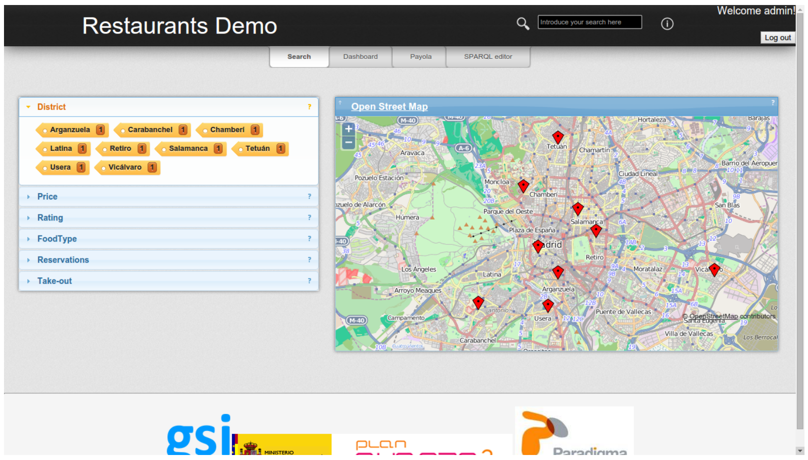


Figure 5.8: Accordion layout

5.3.4 Openlayers and GeoJSON

In this case study, we will not only represent points on a map, but also polygons. The geographical data about districts stored in Fuseki is in GeoSPARQL format, as you can see in[5.5].

We can easily represent it into an Openlayers map by using *OpenLayers.Format.GeoJSON*[9] class. But since that Openlayers requires a specific format GeoJSON for proper representation, and our districts' information is in GeoSPARQL format, we were required to develop the *Geo Proxy [4.4].* to convert GeoSPARQL to GeoJSON. Once the data is converted, we have a *FeatureCollection* containing all districts polygons in the respective features. The next figure shows an example of feature in GeoJSON.

```
{
  "type": "Feature",
  "geometry": {
    "type": "MultiPolygon",
    "coordinates": [  [440414.61, 4473164.38], ..., [440414.61,
       4473164.38]]
  },
  "properties": {
    "s": {},
    "fGeom": {},
    "fWKT": {},
    "geocodigo": {},
    "desbdt": {},
    "dbpediaLink": {},
    "price": {},
    "foodtype": {},
    "stars": {},
    "reservations": {},
    "takeout": {},
    "lat": {},
    "long": {}
  }
}
```

Listing 5.5: Restaurants results JSON example

[9]http://dev.openlayers.org/docs/files/OpenLayers/Format/GeoJSON-js.html

5.4 Slovakian dataset

In this case study we will use the application from a more professional standpoint. For this purpose, we will work with the dataset provided by the Slovak Environmental Agency (SEA[10]) about harmonised protected sites dataset according to INSPIRE[11] Data Specification on Protected Sites – Guidelines through WFS service interface.

In this case we will also work with GeoJSON data as in the case before, so we will test about the same features, but the purpose of this case is to test the performance of the application with a more professional and standardized dataset. In other words, to show a professional GIS application of Sefarad.

5.4.1 Protected sites dataset

SEA provided source files for Geoserver workspaces related to slovak protected sites feature type. Source files can be downloaded from SEA website[12].

Protected sites data are stored in GIS database as simple features in several database tables. Tables slightly differ in structure, they contain some common and some different fields. The structure of database tables is application dependant and was designed in the past during applications development, different applications are used for different kind of protected sites. Upon discussion with domain expert mapping between individual protected sites tables and protected sites designation schemas was established.

Subsequently unifying view could be constructed, unioning several protected sites tables into one logical database view with common set of data fields. Set of common data fields was chosen according to INSPIRE Protected Site simple profile application scheme. Fields for designation scheme and designation scheme value (both mandatory in application schema) were added to view for easing later mapping. Figure 5.9 shows the data scheme.

[10]http://www.sazp.sk/

[11]http://inspire.ec.europa.eu/

[12]http://inspire.geop.sazp.sk/geoserver/www/eenvplus/ps_harmonisation.tgz

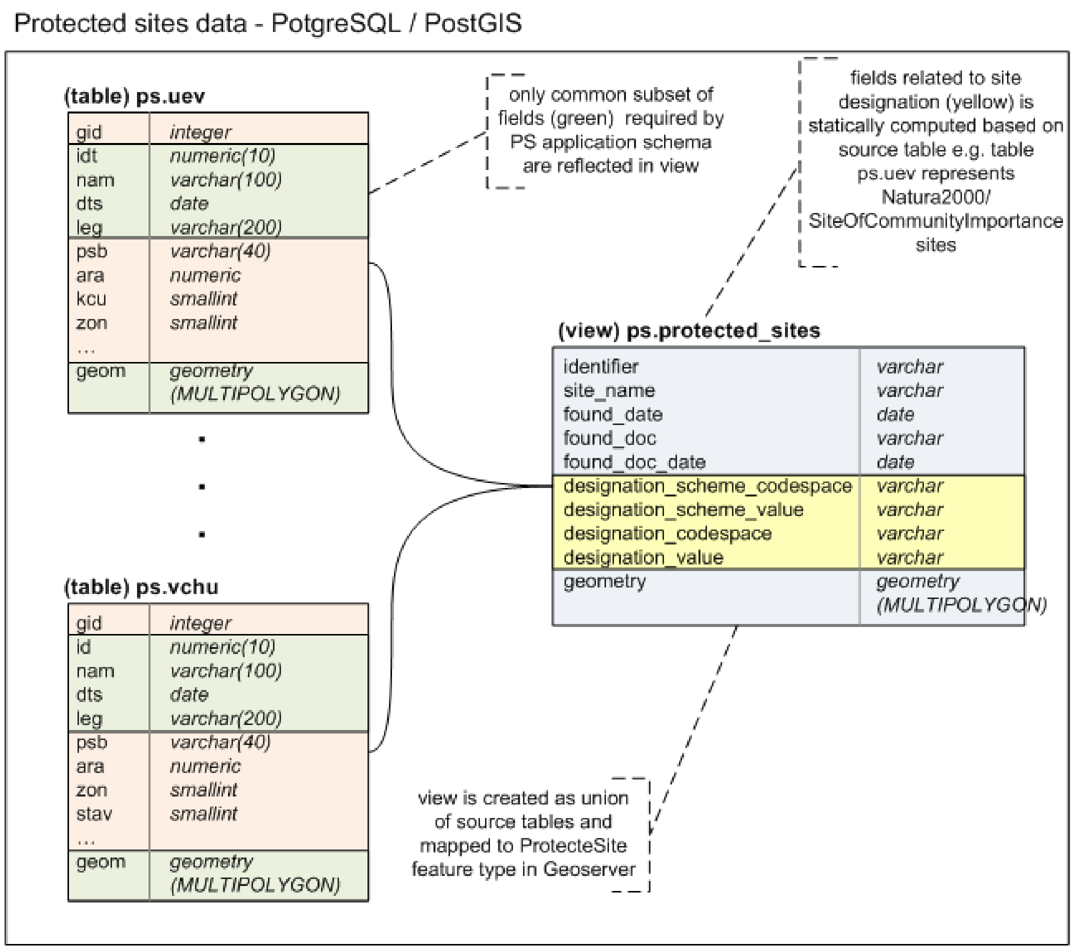

Protected sites data - PotgreSQL / PostGIS
(table) ps.uev
gid integer
idt numeric(10)
nam varchar(100)
dts date
leg varchar(200)
psb varchar(40)
ara numeric
kcu smallint
zon smallint
...
geom geometry (MULTIPOLYGON)
(table) ps.vchu
gid integer
id numeric(10)
nam varchar(100)
dts date
leg varchar(200)
psb varchar(40)
ara numeric
zon smallint
stav smallint
...
geom geometry (MULTIPOLYGON)
only common subset of fields (green) required by PS application schema are reflected in view
fields related to site designation (yellow) is statically computed based on source table e.g. table ps.uev represents Natura2000/ SiteOfCommunityImportance sites
(view) ps.protected_sites
identifier varchar
site_name varchar
found_date date
found_doc varchar
found_doc_date date
designation_scheme_codespace varchar
designation_scheme_value varchar
designation_codespace varchar
designation_value varchar
geometry geometry (MULTIPOLYGON)
view is created as union of source tables and mapped to ProtecteSite feature type in Geoserver

Figure 5.9: Protected sites tables and view

The final data have the following fields and information values, which will be used to test faceted browsing and geo filtering with ECQL[13].

INSPIRE Designation	SK Designation	SK Legislation
natura2000/siteOfCommunity OImportance	Uzemia europskeho vyznamu/ Sites of Community importance	Act of NC SR No. 543/2002 on Nature and Landscape Protection
natura2000/specialProtectionArea	Chranene vtacie uzemia/ Special protection areas	Act of NC SR No. 543/2002 on Nature and Landscape Protection

[13]http://docs.geoserver.org/latest/en/user/filter/ecql_reference.html

ramsar/ramsar	Ramsar/Ramsar sites	Act of NC SR No. 543/2002 on Nature and Landscape Protection
UNESCOWorldHeritage/natural	Unesco/Unesco natural heritage sites	Act of NC SR No. 543/2002 on Nature and Landscape Protection
UNESCOManAndBiosphereProgramme/biosphereReserve	Biosfericke rezervacie/Biosphere reserves	Act of NC SR No. 543/2002 on Nature and Landscape Protection
IUCN/nationalPark	Velkoplošné chránené územia/Large scale protected ares Národný park/National park	Act of NC SR No. 543/2002 on Nature and Landscape Protection
	Velkoplošné chránené územia/Large scale protected ares Chránené krajinné oblasti /Protected landscape area	Act of NC SR No. 543/2002 on Nature and Landscape Protection
IUCN/strictNatureReserve	Maloplošné chránené územia (Small scale protected areas): Prírodná rezervácia/Nature reserve	Act of NC SR No. 543/2002 on Nature and Landscape Protection
	Maloplošné chránené územia (Small scale protected areas): Národná prírodná rezervácia/National nature reserve	Act of NC SR No. 543/2002 on Nature and Landscape Protection

	Maloplošné chránené územia(Small scale protected areas): Ochranné pásmo prírodnej rezervácie/Buffer zone of natural reserve	Act of NC SR No. 543/2002 on Nature and Landscape Protection
	Maloplošné chránené územia(Small scale protected areas): Ochranné pásmo národnej prírodnej rezervácie/Buffer zone of national nature reserve	Act of NC SR No. 543/2002 on Nature and Landscape Protection
IUCN/ naturalMonument	Maloplošné chránené územia(Small scale protected areas): Chránený krajinný prvok/Protected landscape element	Act of NC SR No. 543/2002 on Nature and Landscape Protection
	Maloplošné chránené územia(Small scale protected areas): Prírodná pamiatka/Natural monument	Act of NC SR No. 543/2002 on Nature and Landscape Protection
	Maloplošné chránené územia(Small scale protected areas): Národná prírodná pamiatka/National natural monument	Act of NC SR No. 543/2002 on Nature and Landscape Protection

	Maloplošné chránené územia(Small scale protected areas): Chránený areál/Protected site	Act of NC SR No. 543/2002 on Nature and Landscape Protection
	Maloplošné chránené územia(Small scale protected areas): Ochranné pásmo prírodnejpamiatky/Buffer zone of natural monument	Act of NC SR No. 543/2002 on Nature and Landscape Protection
	Maloplošné chránené územia(Small scale protected areas): Ochranné pásmo národnej prírodnej pamiatky/Buffer zone of national natural monument	Act of NC SR No. 543/2002 on Nature and Landscape Protection
IUCN/ wildernessArea	Maloplošné chránené územia(Small scale protected areas): Ochranné pásmo chráneného areálu/Buffer zone of protected site	Act of NC SR No. 543/2002 on Nature and Landscape Protection
	Chránené krajinné územie (Protected landscape area)	Act of NC SR No. 543/2002 on Nature and Landscape Protection

Table 5.1: Data INSPIRE designation

5.4.2 SPARQL Query and results data

In this case, we also store the information into a local Fuseki database and we query it from Sefarad. Query for retrieving the corresponding information is shown below.

```
PREFIX drf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX j.0: <http://inspire.jrc.ec.europa.eu/schemas/gn/3.0/>
PREFIX j.1: <http://inspire.jrc.ec.europa.eu/schemas/ps/3.0/>
PREFIX j.2: <http://inspire.jrc.ec.europa.eu/schemas/base/3.2/>
PREFIX j.3: <http://www.opengis.net/ont/geosparql#>

SELECT *
  WHERE {
  SERVICE <http://localhost:3030/slovakia/query> {
    ?res j.3:hasGeometry ?fGeom .
    ?fGeom j.3:asWKT ?fWKT .
    ?res j.1:siteProtectionClassification ?spc  .
    ?res j.1:LegalFoundationDate ?lfd .
    ?res j.1:LegalFoundationDocument ?lfdoc .
    ?res j.1:inspireId ?inspire .
    ?inspire j.2:namespace ?namespace .
    ?inspire j.2:namespace ?localId .
    ?res j.1:siteDesignation ?siteDesignation .
    ?siteDesignation j.1:percentageUnderDesignation ?
        percentageUnderDesignation .
    ?siteDesignation j.1:designation ?designation .
    ?siteDesignation j.1:designationScheme ?designationScheme .
  }
}

LIMIT 10;
```

Listing 5.6: Slovakian Demo SPARQL

At this point, a piece of the information returned from the server is as follows.

```
{
  "res": {
    "type": "uri",
    "value": "http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK
        /SKNATS942"
  },
  "fGeom": {
    "type": "uri",
    "value": "http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK
        /geometry/SKNATS942"
  },
  "fWKT": {
    "datatype": "http://www.opengis.net/ont/sf#wktLiteral",
    "type": "typed-literal",
    "value": "MULTIPOLYGON(((17.65642811769707
        48.1686865811456,..., 17.65642811769707 48.1686865811456)))"
  },
  "spc": {
    "type": "literal",
    "value": "natureConservation ecological environment"
  },
  "lfd": {
    "type": "literal",
    "value": ""
  },
  "lfdoc": {
    "type": "literal",
    "value": ""
  },
  "inspire": {
    "type": "uri",
    "value": "http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK
        /inspireId/SKNATS942"
  },
  "namespace": {
    "type": "literal",
    "value": "SK:GOV:MOE:SEA:PS"
  },
  "localId": {
    "type": "literal",
    "value": "SK:GOV:MOE:SEA:PS"
  },
  "siteDesignation": {
    "type": "uri",
    "value": "http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK
        /siteDesignation/SKNATS942"
  },
  "percentageUnderDesignation": {
    "type": "literal",
    "value": ""
  },
  "designation": {
    "type": "literal",
    "value": "wildernessArea"
  },
  "designationScheme": {
    "type": "literal",
    "value": "IUCN"
  }
}
```

Listing 5.7: Slovakia results JSON example

As you can see, JSON results contain many geographical properties, following the data scheme explained in [5.1].

Up to this point, the process is the same as that followed in the previous case study.

5.4.3 OpenStreet Map: GeoJSON representation

In this case study, we will use the features for polygons representation that Openlayers offers. First of all, we need to convert the retrieved geoSPARQL data to GeoJSON data by using the GeoProxy. After doing that, we will have a `FeatureCollection` containing one feature for each parcel in the dataset. An example of one of this features is shown in the next listing.

```
{
  "type": "Feature",
  "geometry": {
    "type": "MultiPolygon",
    "coordinates": [
      [
        [
          [17.65642811769707, 48.1686865811456],
          [...]
          [17.65642811769707, 48.1686865811456]
        ]
      ]
    ]
  },
  "properties": {
  }
}
```

Listing 5.8: Slovakian feature example

Drawing it with Openlayers and OpenStreet Map, we get the following map result.

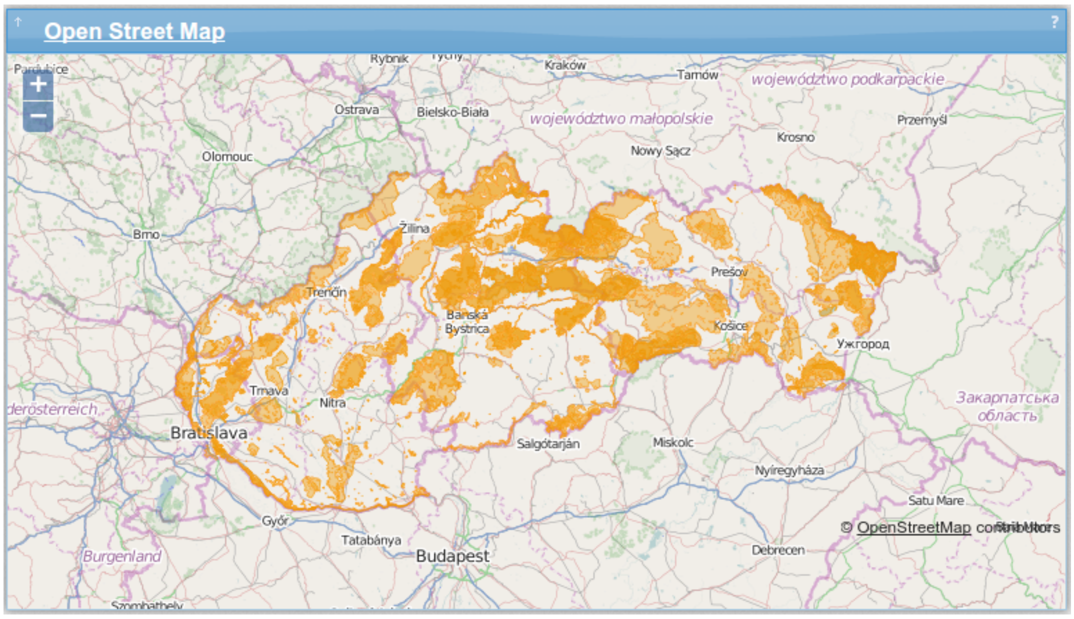
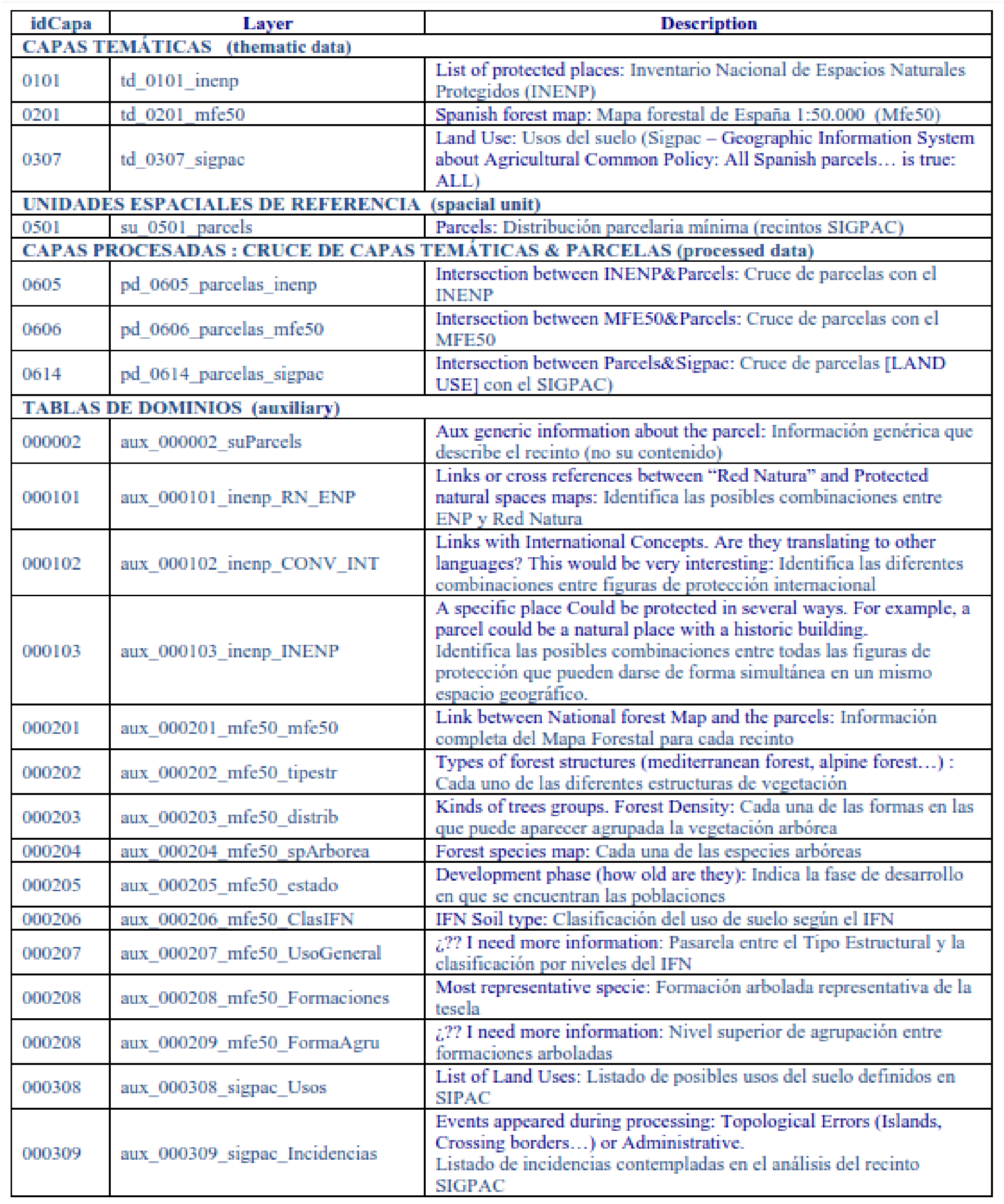


Figure 5.10: Protected sites map

5.5 SmartOpenData parcels dataset

Finally, in this case study we will use a shapefiles dataset provided by *Tragsa*[14] for the SmartOpenData project. The main purpose of this case study is to test Sefarad performance with GeoServer. Tragsa has provided us a new dataset about different kinds of parcels in Spain in DBF and shapefile format. The data scheme is shown in Figure 5.11.



idCapa	Layer	Description
CAPAS TEMÁTICAS (thematic data)		
0101	td_0101_inenp	List of protected places: Inventario Nacional de Espacios Naturales Protegidos (INENP)
0201	td_0201_mfe50	Spanish forest map: Mapa forestal de España 1:50.000 (Mfe50)
0307	td_0307_sigpac	Land Use: Usos del suelo (Sigpac – Geographic Information System about Agricultural Common Policy: All Spanish parcels… is true: ALL)
UNIDADES ESPACIALES DE REFERENCIA (spacial unit)		
0501	su_0501_parcels	Parcels: Distribución parcelaria mínima (recintos SIGPAC)
CAPAS PROCESADAS : CRUCE DE CAPAS TEMÁTICAS & PARCELAS (processed data)		
0605	pd_0605_parcelas_inenp	Intersection between INENP&Parcels: Cruce de parcelas con el INENP
0606	pd_0606_parcelas_mfe50	Intersection between MFE50&Parcels: Cruce de parcelas con el MFE50
0614	pd_0614_parcelas_sigpac	Intersection between Parcels&Sigpac: Cruce de parcelas [LAND USE] con el SIGPAC)
TABLAS DE DOMINIOS (auxiliary)		
000002	aux_000002_suParcels	Aux generic information about the parcel: Información genérica que describe el recinto (no su contenido)
000101	aux_000101_inenp_RN_ENP	Links or cross references between "Red Natura" and Protected natural spaces maps: Identifica las posibles combinaciones entre ENP y Red Natura
000102	aux_000102_inenp_CONV_INT	Links with International Concepts. Are they translating to other languages? This would be very interesting: Identifica las diferentes combinaciones entre figuras de protección internacional
000103	aux_000103_inenp_INENP	A specific place Could be protected in several ways. For example, a parcel could be a natural place with a historic building. Identifica las posibles combinaciones entre todas las figuras de protección que pueden darse de forma simultánea en un mismo espacio geográfico.
000201	aux_000201_mfe50_mfe50	Link between National forest Map and the parcels: Información completa del Mapa Forestal para cada recinto
000202	aux_000202_mfe50_tipestr	Types of forest structures (mediterranean forest, alpine forest…) : Cada uno de las diferentes estructuras de vegetación
000203	aux_000203_mfe50_distrib	Kinds of trees groups. Forest Density: Cada una de las formas en las que puede aparecer agrupada la vegetación arbórea
000204	aux_000204_mfe50_spArborea	Forest species map: Cada una de las especies arbóreas
000205	aux_000205_mfe50_estado	Development phase (how old are they): Indica la fase de desarrollo en que se encuentran las poblaciones
000206	aux_000206_mfe50_ClasIFN	IFN Soil type: Clasificación del uso de suelo según el IFN
000207	aux_000207_mfe50_UsoGeneral	¿?? I need more information: Pasarela entre el Tipo Estructural y la clasificación por niveles del IFN
000208	aux_000208_mfe50_Formaciones	Most representative specie: Formación arbolada representativa de la tesela
000208	aux_000209_mfe50_FormaAgru	¿?? I need more information: Nivel superior de agrupación entre formaciones arboladas
000308	aux_000308_sigpac_Usos	List of Land Uses: Listado de posibles usos del suelo definidos en SIPAC
000309	aux_000309_sigpac_Incidencias	Events appeared during processing: Topological Errors (Islands, Crossing borders…) or Administrative. Listado de incidencias contempladas en el análisis del recinto SIGPAC

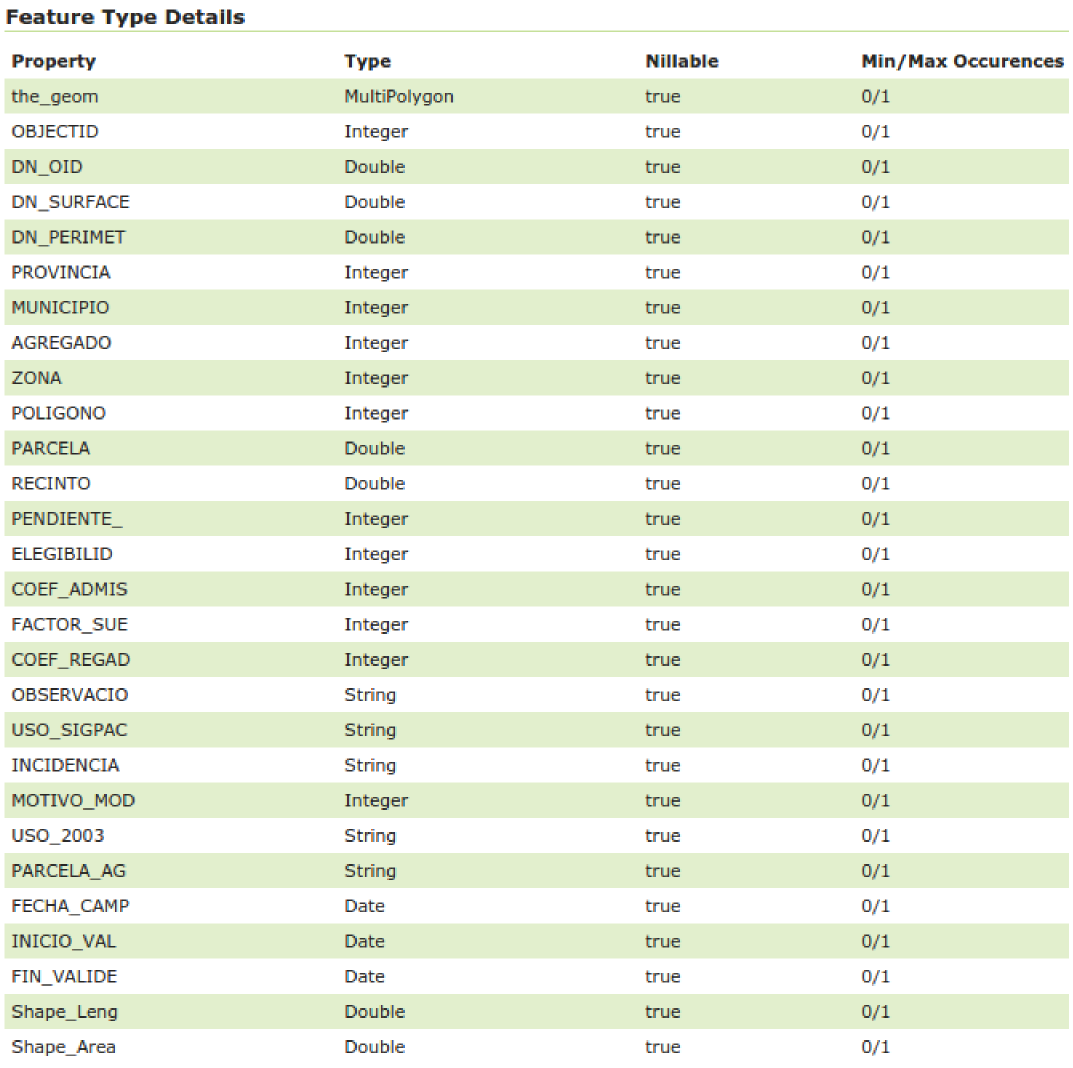
Figure 5.11: Dataset scheme

[14] http://www.tragsa.es/

5.5.1 Parcels data scheme

We have stored all the shapefiles in our local GeoServer installation. From there, we are able to see all the different parcels, see its feature types, preview them with Openlayers directly from GeoServer, test ECQL filtering, etcetera.

For this case study, we have chosen the `td_0307_sigpac` parcels, whose feature types are detailed in Figure 5.12.



Feature Type Details

Property	Type	Nillable	Min/Max Occurences
the_geom	MultiPolygon	true	0/1
OBJECTID	Integer	true	0/1
DN_OID	Double	true	0/1
DN_SURFACE	Double	true	0/1
DN_PERIMET	Double	true	0/1
PROVINCIA	Integer	true	0/1
MUNICIPIO	Integer	true	0/1
AGREGADO	Integer	true	0/1
ZONA	Integer	true	0/1
POLIGONO	Integer	true	0/1
PARCELA	Double	true	0/1
RECINTO	Double	true	0/1
PENDIENTE_	Integer	true	0/1
ELEGIBILID	Integer	true	0/1
COEF_ADMIS	Integer	true	0/1
FACTOR_SUE	Integer	true	0/1
COEF_REGAD	Integer	true	0/1
OBSERVACIO	String	true	0/1
USO_SIGPAC	String	true	0/1
INCIDENCIA	String	true	0/1
MOTIVO_MOD	Integer	true	0/1
USO_2003	String	true	0/1
PARCELA_AG	String	true	0/1
FECHA_CAMP	Date	true	0/1
INICIO_VAL	Date	true	0/1
FIN_VALIDE	Date	true	0/1
Shape_Leng	Double	true	0/1
Shape_Area	Double	true	0/1

Figure 5.12: Feature Types Details

5.5.2 GeoServer and Openlayers

In this case, we do not need the GeoProxy to convert the retrieved data. We can query to GeoServer directly from Sefarad and represent the geographical data in an Openlayers map. For this case, the SPARQL query is shown in Listing 5.9.

```
PREFIX ns3:   <http://smartod.gsi.dit.upm.es/parcels0307/schema#>

SELECT ?shape_area ?parcel ?slope ?use WHERE {
      ?s a ns3:Parcels .
      ?s a ?itemtype ;
      ns3:Shape_Area ?shape_area ;
      ns3:PENDIENTE_ ?slope ;
      ns3:USO_SIGPAC ?use ;
      ns3:PARCELA ?parcel
}
```

Listing 5.9: SMOData Dataset SPARQL query

We must also indicate the graph of the requested data, which is: `http://dataset1.smartopen.gsi.edu`.

After receiving the results, we can directly represent them without any conversion in an Openlayers map, as shown in Figure 5.13.

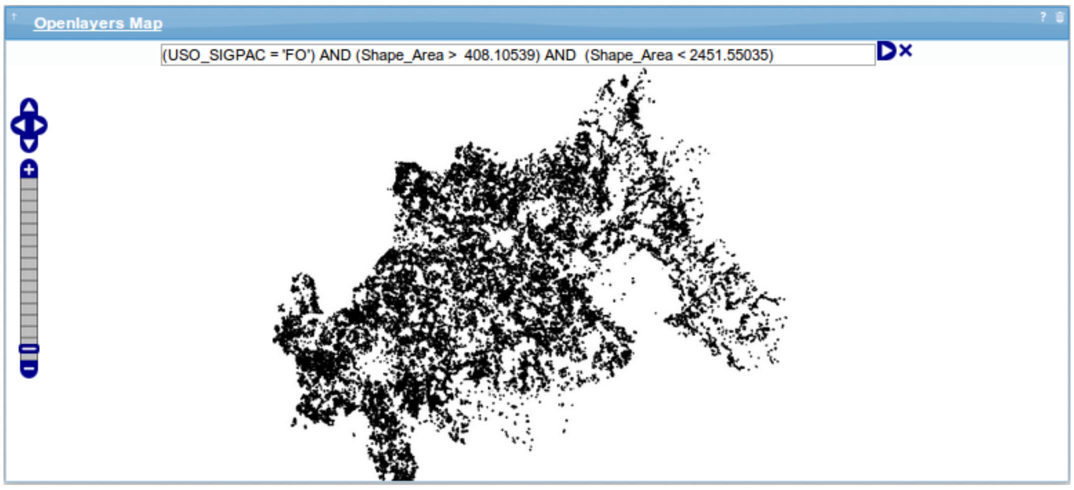
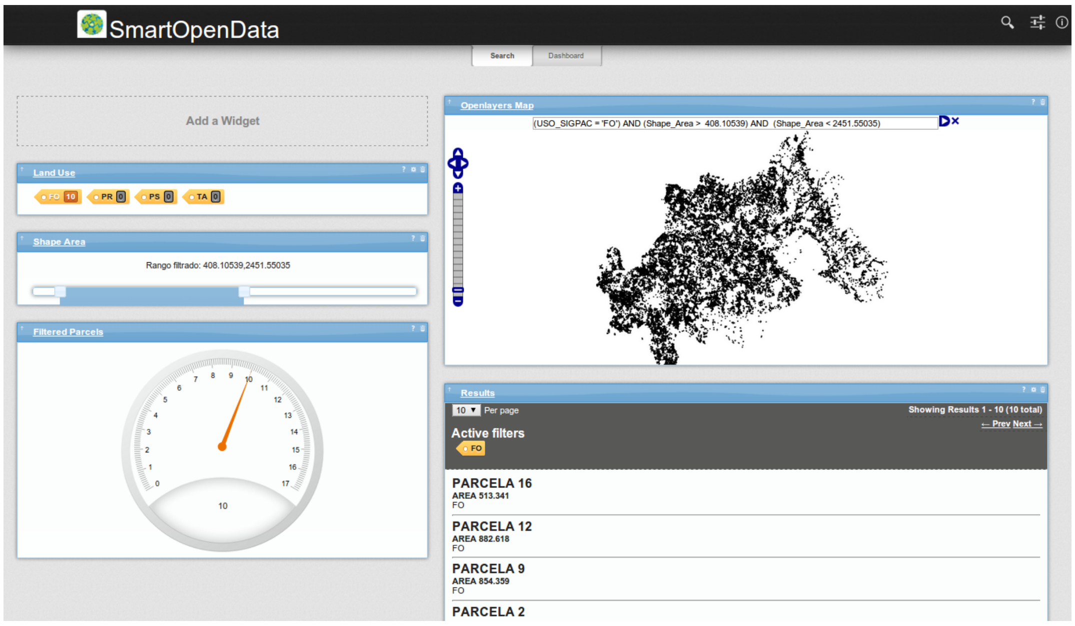

Openlayers Map
(USO_SIGPAC = 'FO') AND (Shape_Area > 408.10539) AND (Shape_Area < 2451.55035)

Figure 5.13: Openlayers map

5.5.3 Geo-filtering: ECQL

As you can see in Figure 5.13, this widget includes a filtering box which allows us to test geofiltering with the retrieved data. We can include any ECQL query in this field and the geofiltering module will automatically update and rerun the query to receive the information that meets the new criteria. Figure 5.14 shows the complete layout of this case study.



(USO_SIGPAC = 'FO') AND (Shape_Area > 408.10539) AND (Shape_Area < 2451.55035)

Rango filtrado: 408.10539,2451.55035

Showing Results 1 - 10 (10 total)

PARCELA 16
AREA 513.341
FO

PARCELA 12
AREA 882.618
FO

PARCELA 9
AREA 854.359
FO

PARCELA 2

Figure 5.14: SmartOpenData layout

CHAPTER 6

Conclusions and future lines

This chapter summarizes the conclusions extracted from this master thesis and the objectives achieved are evaluated. After that, we describe thinkings about future work.

6.1 Project outcomes

The main outcome of this project is the developed system. By extending the existing Sefarad framework with the suitable modules and widgets, we have created a powerful querying and visualization framework for Geo Linked Data, enabling final user to query any dataset he wants and manage, index, sort and filter the retrieved data.

Due to integration with different types of databases as data sources, as explained in Section 4.5, the user is able to store data in many formats and query, filter and represent it from Sefarad. Furthermore, the user can query any semantic endpoint due to the compatibility with *Semantic Web* and *Linked Data* specifications. Moreover, the *GeoProxy* (Section 4.4) solves format compatibility issues for representation with Openlayers.

Otherwise, the development of the *User Management Module*, detailed in Section 4.8, provides multiple important features. On the one hand, this module allows the users to save and load their own preferences and settings each time they use Sefarad, which makes the application more configurable and customizable. In the other hand, the application administrator can use this module to manage the information (credentials, permissions, etcetera) about the different users. Moreover, the integration with MongoDB can be exploited to store large amounts of data in JSON format thereby improving application performance.

The introduction of geo filtering capabilities, besides to faceted browsing and keyword search (Section 4.6), enables new powerful filtering options for geographic data. By using CQL language, the user can write filtering queries with a familiar text-based syntax, which is thus more readable and better-suited for manual authoring.

To make the application more responsive and appropriate to filter data with many fields of information, we have developed the *accordion layout* described in Section 4.7.2.2. This provides the user a cleaner and more usable interface, making it easier to display filtering and representation widgets on the same screen.

Finally, the inclusion of the Grunt.js task runner (Section 4.9.2) and the reorganization of the project source files facilitate the work of developers. Now, any developer can create a new widget just by using the widgets template and running the corresponding Grunt command, as explained in Appendix B.

All these features has been tested in the different case studies (Section 5). These experiments have allowed us to verify the correct operation of the system and all its components and realize some improvements needed for a better user experience.

6.2 Achieved goals

In Chapter 1 we mentioned a list of goals for the project. The achieved goals can be summarised as follows:

Analyse the state of the Semantic Web technologies and study all related standards. This goal has been achieved successfully. Its results are presented in Chapter 2, where we analyse and describe the state of the art and evaluate the need for taking advantage of the benefits that the Semantic Web provides to the geographical area.

Study the different web technologies that will help us to develop the application. This goal has been achieved successfully. Its results are also presented in Chapter 2, where we study and analyze the usefulness in the different facets of the project and present the different technologies that have helped us to develop this framework.

Determine the architecture of the application. This goal has been achieved successfully. The complete architecture of the system and a detailed explanation of all its modules and submodules is included in Chapter 4. We have defined a modular structure for the project to be more maintainable and extensible.

Develop each one of the modules that make up the system. This goal has been achieved successfully. We have explained the operation of all modules in Chapter 4.

Test the application on different case studies. This goal has been achieved successfully. We have test the application on four different case studies, detailed in Chapter 5. We have tested the application in different areas, trying to test all functions developed to determine possible bugs, possible improvements or guarantee proper operation.

Document the work done for future users and developers. This goal has been achieved successfully. For this purpose, we have included two appendices at the end of this master thesis. The users can also visit the wiki[1] of the project for extra information.

6.3 Conclusions

This project has allowed us to perform a deep insight into the *Semantic Web* and *Linked Data* and all its benefits, and specially into the way of applying these techniques to the geographic scope.

This project has been developed in the scope of the SmartOpenData project contributing

[1]https://github.com/gsi-upm/sefarad/wiki

to this European project, so we have worked with companies such as Tragsa[2]. To work in a working group has helped us to organize tasks and responsibilities and has forced us to organize with our partners. It is always a good source of learning.

We have used existing advanced technologies whenever it was possible, and put it together to build a solid and functional system. As an example, we have use MongoDB, Gruntjs, Openlayers, etcetera.

Dividing the project into different modules forced us to rely on existing web and software standards which helped us to integrate and interconnect all of them. Modular architecture also facilitates project development and maintenance.

We experienced big changes as early technology adopters, such as new versions of the GeoSPARQL language and GeoJSON fixing bugs and creating new functionalities. We have found the need to test the tool in order to find failures and possible improvements. Some of our modules and developments are the result of experimentation and detection of new needs.

6.4 Future work

The project outcome can also serve as a solid base for future work and development. In the following points some fields of study or improvement are presented to continue the development.

- Enable Sefarad to handle heterogeneous data collections, so that different queries can be performed, storing multiple data types and handle them separately or combined.
- Improve the performance of the application when large amounts of data are queried, so the service get not blocked or later an excessively long time to respond.
- Integrate SirenDB as a local database, in order to improve the performance of the application with large amounts of data and take advantage of SOLR indexing.
- Improve geographical search operators, such as intersections between polygons or polygons contained within other polygons.
- Integrate Quepy[3] Framework, to allow non-technical users to execute querires in natural language.

[2]http://www.tragsa.es/

[3]http://quepy.readthedocs.org/en/latest/index.html

- Improve the installer. It has been developed a first version of the installer,but it should be improved so that it installs everything needed for the execution of the application, such as MongoDB, Grunt.js, etcetera.

APPENDIX A

Installing and configuring Sefarad

This tutorial goes through the process of installing and configuring Sefarad in any computer with a Linux OS. Project's code is available at https://github.com/gsi-upm/demo-smartopendata. After the installation, the user will be able to run some predefined demos or run the application itself and to modify the source files in order to introduce his own changes.

A.1 Installation

A.1.1 Requirements

- Node.js versions [>= 0.8.0][1]
- Grunt.js[2]
- MongoDB[3]
- MondoDB PHP Driver[4]
- The next npm devDependencies listed below should be installed. In linux, this can be done with a simple command in the project folder `sudo npm install`
  - grunt-contrib-concat
  - grunt-processhtml
  - grunt-contrib-copy
  - grunt-contrib-clean

A.1.2 Installation steps

- Install Node.js versions [>= 0.8.0] following its installation guide.
- Install MongoDB following the manual and start it with:

```
sudo service mongod start
```

- Install Mongo driver for PHP:

```
sudo pecl install mongo
```

- After that, add the next line to the `php.ini` file:

```
extension=mongo.so
```

- Now, you can directly download the .zip folder from GitHub[5] or clone the repository:

[1] http://nodejs.org/
[2] http://gruntjs.com/getting-started
[3] http://docs.mongodb.org/manual/
[4] http://php.net/manual/es/book.mongo.php
[5] https://github.com/gsi-upm/demo-smartopendata

```
cd desired/folder
git clone https://github.com/gsi-upm/demo-smartopendata.git
```

ONLY FOR DEVELOPERS

- Install Grunt's command line interface (CLI), in order to be able to execute grunt commands from the terminal:

```
sudo npm install -g grunt-cli
```

- Finally, you must install needed dependencies

```
cd to/project/folder
sudo npm install
```

After that, the user will have everything needed installed on his computer to run the application. Inside the folder build/, the user will have a fully built version of the project ready for execution. Meanwhile, developers will have all source files in the src/ folder, and using Grunt task runner, will be able to build a new project with updated changes.

APPENDIX B

User Manual

This user manual goes through the most important features for both users and developers. Project's code is available at https://github.com/gsi-upm/demo-smartopendata.

B.1 Create new widget

The great potential of Sefarad for our project lies in the capability to create our own widgets really easily. We should not worry about obtaining the filtered data and updating the widget when a new facet is selected thanks to Knockout framework. For this purpose, the application specifies how to create a new Javascript file in which it should be placed a Javascript object using D3.js[1] framework. The widget template is as follows:

```javascript
var newWidget = {
    name: "Name",
    description: "description",
    img: "path/to/image",
    type: "type",
    help: "help",
    // Category of the widget (1: textFilter, 2: numericFilter, 3: graph,
        5:results, 4: other, 6:map)
    cat: X,

    render: function() {
        var id = 'A' + Math.floor(Math.random() * 10001);
        var field = newWidget.field || "";
        vm.activeWidgetsRight.push({
            "id": ko.observable(id),
            "title": ko.observable(newWidget.name),
            "type": ko.observable(newWidget.type),
            "field": ko.observable(field),
            "collapsed": ko.observable(false),
            "showWidgetHelp": ko.observable(false),
            "help": ko.observable(newWidget.help)
        });
        newWidget.paint(id);
    },

    paint: function(id) {
        d3.select('#' + id).selectAll('div').remove();
        var div = d3.select('#' + id);
        div.attr("align", "center");
    }
};
```

Listing B.1: "Sefarad widget template"

[1] http://d3js.org/

Bibliography

[1] T. Berners-Lee, “Linked data - design issues. w3c.,” 2006.

[2] T. K. Werner Kuhn and K. Janowicz, “Linked data - a paradigm shift for geographic information science.,” 2010.

[3] F. J. López-Pellicer, M. J. Silva, M. S. Chaves, F. J. Zarazaga-Soria, and P. R. Muro-Medrano, “Geo linked data.,” in *DEXA (1)*, pp. 495–502, 2010.

[4] R. Bermejo, “Desarrollo de un Framework HTML5 de Visualización y Consulta Semántica de Repositorios RDF,” Master’s thesis, Universidad Politécnica de Madrid, ETSI Telecomunicación, June 2014.

[5] C. Bizer, T. Heath, and T. Berners-Lees, “Linked data - the story so far.,” in *Special Issue on Linked Data, International Journal on Semantic Web and Information Systems*, pp. 1–22, 2009.

[6] B. Youngblood and S. Iacovella, *GeoServer Beginner’s Guide.* Packt Publishing, 2 2013.

[7] P. A. Santiago, *OpenLayers Cookbook.* Packt Publishing, 8 2012.

[8] A. D. Lorenzo and G. Allegri, *Instant OpenLayers Starter.* Packt Publishing, 4 2013.

[9] B. P. Hogan, *Automate with Grunt: The Build Tool for JavaScript.* Pragmatic Bookshelf, 1 ed., 5 2014.

[10] R. Islam, *PHP and MongoDB Web Development Beginner’s Guide.* Packt Publishing, 11 2011.