PROYECTO FIN DE CARRERA

Título:	Development of a framework for GeoLinked Data query and visualization based on web components
Autor:	Alejandro Saura Villanueva
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

DEVELOPMENT OF A FRAMEWORK FOR GEOLINKED DATA QUERY AND VISUALIZATION BASED ON WEB COMPONENTS

Alejandro Saura Villanueva

Junio de 2015

Resumen

Este documento contiene la descripción del proyecto fin de carrera enfocado en el desarrollo de Sefarad-2.0. Dicho sistema es una aplicación web de consulta y visualización de datos semánticos.

En primer lugar, analizamos el estado actual de la web semántica y el gran crecimiento que está experimentando en la actualidad. Introduciremos la necesidad de desarrollar un software que nos permita consultar los datos de dicha web semántica y tratar de indexarlos, filtrarlos y presentarlos.

Partimos de Sefarad, propiedad del *Grupo de sistemas inteligentes* (GSI[1]). Analizamos sus puntos fuertes y sus carencias y justificamos el desarrollo de esta nueva versión.

Presentamos y justificamos la elección de las tecnologías, teniendo como resultado un conjunto de tecnologías nuevas como *Dart* y*Polymer* que aseguran una compatibilidad con futuros paradigmas web.

Desarrollamos un estudio detallado de la arquitectura de nuestro proyecto final, así como de cada uno de los módulos que lo componen.

Repasamos y analizamos las distintas fases de prototipado por las que ha pasado el proyecto, adjuntando pruebas comparativas de rendimiento y especificando su funcionamiento, arquitectura y posibles vías alternativas. Posteriormente, desarrollamos la descripción de la arquitectura y el funcionamiento del sistema final.

Aplicamos este sistema a varios casos de uso práctico. El más importante de ellos es *SmartOpenData*, para el que construimos widgets y peticiones SPARQL para lograr un resultado similar al obtenido en trabajos anteriores con el framework Sefarad 1.0.

Finalmente, presentamos las conclusiones del proyecto realizado y discutimos posibles trabajos futuros y oportunidades de ampliación y aplicación.

Palabras clave: Tecnologías Semánticas, Sefarad-2.0, RDF, SPARQL, Dart, Angular.js, Polymer, Crossfilter

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

Abstract

This work contains the description of a master thesis focused on the development of Sefarad-2.0. This system is a web based, semantic data browsing and visualization application.

In first place, we analyse the semantic web actual state and its actual growth. We introduce the need of a software that enables to consult and index its data as well as filter and render them.

Our starting point is Sefarad 1.0, property of the *Intelligent Systems Group* GSI[2]. We analyse its strengths and weaknesses, justifying the development of a new version.

We also present the studied technologies and justify their choice, taking as result a compilation of bleeding edge technologies as *Dart* and *Polymer* that assure future developments compatibility.

We describe in depth the architecture of our final product, as well as we study each of its submodules.

We list and test each prototype step we have take in this project, providing benchmark results, comparing their features, describing their operation, architecture and possible development alternatives. After that we present the description of the features and architecture of the final system.

We apply this system to various practical case studies, being the most important between them *SmartOpenData*, for which we will develop some concrete widgets and SPARQL queries so we get an equivalent but improved result to the one achieved with the previous Sefarad 1.0 framework.

Finally, we present the project conclusions and discuss about future works and possible applications and extensions.

Keywords: Semantic technologies, Sefarad-2.0, RDF, SPARQL, Dart, Angular.js, Polymer, Crossfilter

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

Agradecimientos

En primer lugar, gracias a mis padres por todos estos años de apoyo incondicional. Gracias a las oportunidades y posibilidades que me han brindado he podido llegar a terminar esta carrera y, en última instancia, este proyecto.

Gracias también a mis amigos por estos años de camino compartido, por no haberme dejado sentir solo nunca. Gracias a mi novia por estar durante este tiempo a mi lado, por todo su apoyo.

Dar las gracias también a mis compañeros de proyecto y de laboratorio, que han construído un ambiente de trabajo increíble en el que ha sido muy facil integrarse. Ha sido una experiencia de trabajo increíble.

Por último, me gustaría agradecer a Carlos, mi tutor, la confianza que ha depositado en mi durante este tiempo y su guía a lo largo de este proyecto.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This chapter provides a conceptual introduction to the context and challenges of this project besides a brief description of the state of the art. We will set up the goals and the procedures that will be followed through all stages. Finally we describe further the structure of this document, describing each chapter and its content.

1.1 Context

Since the concept of the Semantic Web [1] appeared, we have observed an important growth in numbers and quality of applications that demonstrate the possibilities it has in different areas.

Concretely, some initiatives have been developed around the inclusion of geospatial data in the semantic web. This is the case of GeoLinkedData [2], a Spanish project that make geospatial data from spain available in the form of a RDF [3] dataset, following the INSPIRE [4] directive.

SmartOpenData [5] is another attempt to create a sustainable Linked Open Data infrastructure to promote environmental protection sharing data among public bodies in the european union. It aims to demonstrate the impact of sharing this information from many varied resources developing demonstrators that will provide high quality results in regional development working with semantically integrated resources. In the course of the project we will contribute to this project facing a case study based on a SmartOpenData dataset.

Despite the fact of this growth of available linked data, the user-friendly browsing of this data at all their different facets remains on a dark spot. There is a need of visualization tools that allows the user to browse that immense data sea.

In this context there are a number of technologies that attempt to solve this problem as Payola [6] or redash.io [7]), both of them attempting to provide a intuitive interface between users and datasets and offering some graphic tools to represent these data.

We have studied the workflow of these two platforms besides Sefarad's and set them as our main references in the process of designing our own interface.

1.2 Master thesis goals

The goal of this Master Thesis is to develop a web application that allows a non-technical user to query geospatial linked data, retrieve and filter the results and make use of graphical tools to extract useful information.

We present Sefarad, a web-based data visualization and browsing application. Sefarad can be used to define, execute and visualize queries to different endpoints. The feature that puts Sefarad apart is its analysis dashboard, capable of faceted search and map render thousands of data thanks to the power of bleeding edge technologies as web components [8] and crossfilter [9], as well as its focus on non-technical users, who are capable of analysing datasets through our template queries system.

In order to achieve this, we will face these challenges:

- Study and test different web technologies that could help us to develop the application, reaching conclusions for each one under certain criteria.
- Design the Architecture of the application through prototype iteration.
- Compare and document the features of each iteration, benchmarking where possible.
- Develop one or more case studies to test the final application and demonstrate its possibilities.
- Document the final application to ease future developments or use cases.

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 context of the project, introducing concepts like Semantic web and Linked Data. After that, we explain the goals of this master thesis and provide the structural overview of this document.

Chapter 2: lists the main technologies that contribute in our project and justify their use based on their advantages.

Chapter 3: describes in depth all the important aspect of Sefarad 2.0, our product, Defining its architecture, its interaction model and reviewing all the widgets that we have develop for it. It also gives a guideline to develop our own dashboard.

Chapter 4: describes the three case studies that we have faced in the project, detailing for each one the origin of the data, how we process it and which analysis model have we used to render that data.

Chapter 5: is a comparison of every stage the project has passed through, detailing for each one the new additions, giving comparative measures and conclusions about what works and what doesn't, as well as arising problems and ideas of how to solve them.

CHAPTER 2

Enabling Technologies

This chapter introduces which technologies have made possible this project. First of all we must introduce Linked Data, RDF, SPARQL and GeoLinked Data [10, 11]. Then we'll move on presenting all technologies that enable us to build a semantic front end with a data filtering and rendering system.

2.1 Linked Data

Linked data is an attempt to describe entities, its properties and relationship with the objective of making them easily treated by machines. Linked Data has been recently suggested as one of the best alternatives for creating these shared information spaces [12]. It 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[1] (also called Web of Data).

This master thesis aims to create a web application that enables a non-technical user to use the Linked Data Web. We will provide a framework where querying Linked Data and Geo Linked Data and visualizing interactively the results through dashboards will be possible.

2.1.1 RDF

Resource Description Framework (RDF) uses URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a “triple”). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications.

Below, a sample RDF/XML file is shown, listing a table with two records and three fields:

```
<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ANIMALS="http://www.some-fictitious-zoo.com/rdf#">

  <RDF:Seq about="http://www.some-fictitious-zoo.com/all-animals">
    <RDF:li>
       <RDF:Description about="http://www.some-fictitious-zoo.com/mammals/lion">
         <ANIMALS:name>Lion</ANIMALS:name>
         <ANIMALS:species>Panthera leo</ANIMALS:species>
         <ANIMALS:class>Mammal</ANIMALS:class>
       </RDF:Description>
    </RDF:li>
    <RDF:li>
       <RDF:Description about="http://www.some-fictitious-zoo.com/mammals/hippopotamus">
         <ANIMALS:name>Hippopotamus</ANIMALS:name>
         <ANIMALS:species>Hippopotamus amphibius</ANIMALS:species>
         <ANIMALS:class>Mammal</ANIMALS:class>
       </RDF:Description>
    </RDF:li>
  </RDF:Seq>
</RDF:RDF>
```

Listing 2.1: RDF/XML document example

[1]http://www.w3.org/standards/semanticweb/

Each RDF:Description tag describes a single record. Within each record, three fields are described, name, species and class. Each of three fields have been given a namespace of ANIMALS, the URL of which has been declared on the RDF tag, where the semantic schema is defined.

The Linked Data paradigm hides the complexity of conceptual databases, maintaining them internal to the data providers and offering an interface where the user only has to know the semantics occurring in the data, the types that can conform subject-predicate expressions as triples in RDF form. This focus developers on specifying and sharing vocabularies describing their data instead of granting access to complex distributed databases.

2.1.2 SPARQL

Linked Data can be queried using SPARQL[2] (an acronym for SPARQL Protocol and RDF Query Language), a query language for RDF which became an official W3C Recommendation[3]. 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. Results of SPARQL queries can be presented in several different forms, such as JSON, RDF/XML, etc.

Here we present an example of a SPARQL query done against dbpedia, one of the largest endpoints available online:

```
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbpprop: <http://dbpedia.org/property/>
PREFIX dbres: <http://dbpedia.org/resource/>

SELECT ?y WHERE {
 ?y dbpedia-owl:binomialAuthority dbres:
    Johan_Christian_Fabricius.
 }

limit 10
```

Listing 2.2: SPARQL query example

[2]http://www.w3.org/TR/rdf-sparql-query
[3]http://www.w3.org/blog/SW/2008/01/15/

In this query, we import the dbpedia semantic schema and look for entities that match with our triples conditions. Please note how this is done through semantic statements instead of tables exploration.

2.1.3 Geo Linked Data

In the geospatial context, GeoLinked Data[4] is an open initiative whose aim is to enrich the Semantic Web with geospatial data into the context of INSPIRE[5] *(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.

GeoSPARQL[6] defines a vocabulary for representing geospatial data in RDF, and it defines an extension to the SPARQL query language for processing geospatial data.

An example of these extra spatial relations is listed in 2.3.

```
PREFIX spatial:<http://jena.apache.org/spatial#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX gn:<http://www.geonames.org/ontology#>

Select *
WHERE{
?object spatial:nearby(40.74 -73.989 1     ).
?object rdfs:label ?label
}LIMIT 10
```

Listing 2.3: geoSPARQL query example

[4]http://linkedgeodata.org/
[5]http://inspire.ec.europa.eu/
[6]http://www.opengeospatial.org/standards/geosparql

2.2 Web Components

Web Components [8] are a set of standards currently being produced by Google engineers as a W3C specification that allows for the creation of reusable widgets or components in web documents and web applications. The intention behind them is to bring component-based software engineering to the World Wide Web. The components model enables encapsulation and interoperability of individual HTML elements.

This idea comes from the union of four main standards: custom HTML elements, HTML imports, templates and shadow DOMs.

2.2.1 Custom HTML Elements

Custom Elements let the user define his own element types with custom tag names. JavaScript code is associated with the custom tags and uses them as an standard tag. That code gets executed each time the compiler reads that tag.

Standard DOM methods can be used on Custom Elements, as accessing their properties, attaching event listeners, and styling them using CSS as with any standard tag.

Thanks to custom tags the amount of code is reduced, internal details encapsulated, APIs per element type can be implemented, productivity is increased by reusing elements and advantage of inheritance is taken to develop new tags based on existing ones.

2.2.2 HTML Imports

HTML Imports let users include and reuse HTML documents in other HTML documents, as 'script' tags let include external Javascript in pages. In particular, these imports include custom element definitions from external URLs. HTML imports use the import relation on a standard 'link' tag.

2.2.3 Templates

Templates define a new 'template' element which describes a standard DOM-based approach for client-side templating. Templates allow developers to declare fragments of markup which are parsed as HTML, go unused at page load, but can be instantiated later on at runtime.

2.2.4 Shadow DOM

There is a fundamental problem that makes widgets built out of HTML and JavaScript hard to use: the DOM tree inside a widget isn't encapsulated from the rest of the page. This lack of encapsulation means that the document stylesheet might accidentally be applied to parts inside the widget; JavaScript might accidentally modify parts inside the widget; IDs might overlap with IDs inside the widget; and so on.

Shadow DOM separates content from presentation therefore eliminating naming conflicts and improving code expression. It is internal to the element and hidden from the end-user.

2.3 Polymer

Polymer[7] is an implementation o these four technologies in one elegant framework of constructing web-components.

Polymer makes it simple to create web components, declaratively. Custom elements are defined using our custom element, 'polymer-element', and can leverage Polymer's special features. These features reduce boilerplate and make it even easier to build complex, web component-based applications:

- Two-way data binding: Data binding extends HTML and the DOM APIs to support a sensible separation between the UI (DOM) of an application and its underlying data (model). Updates to the model are reflected in the DOM and user input into the DOM is immediately assigned to the model.

- Declarative event handling: Binding of events to methods in the component. It uses special on-event syntax to trigger this binding behavior.

- Declarative inheritance: A Polymer element can extend another element by using the extends attribute. The parent's properties and methods are inherited by the child element and data-bound.

- Property observation: All properties on Polymer elements can be watched for changes by implementing a propertyNameChanged handler. When the value of a watched property changes, the appropriate change handler is automatically invoked.

[7]https://www.polymer-project.org/0.5/

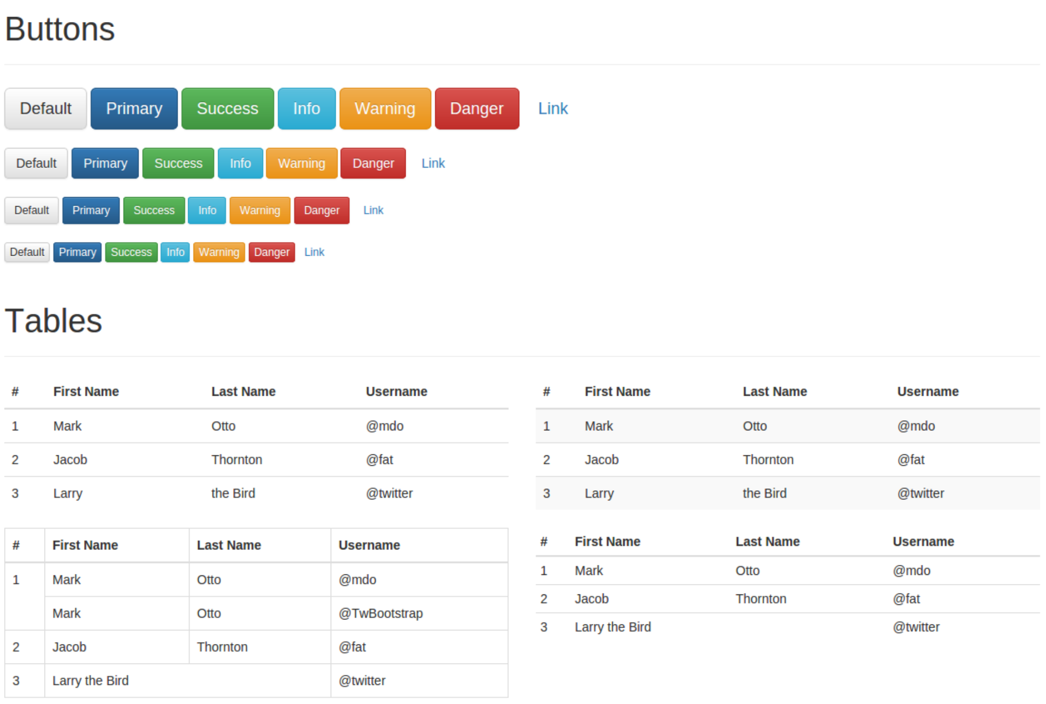
2.4 Client-side web technologies

2.4.1 Bootstrap

Bootstrap[8] is the most popular HTML, CSS, and JavaScript framework for developing responsive web sites. It features a multidevice preprocessor, based in Less[9] and Sass[10] that provides instant coherent style to the most common HTML elements and CSS components as tables, buttons, text fields, etc.

It offers a simple but powerful interface. All we have to do to integrate bootstrap styled elements into our website is add key class names to the supported elements and bootstrap will do the rest.

An example applied to HTML buttons and tables tables:


Buttons
Default Primary Success Info Warning Danger Link
Default Primary Success Info Warning Danger Link
Default Primary Success Info Warning Danger Link
Default Primary Success Info Warning Danger Link
Tables

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
	Mark	Otto	@TwBootstrap
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

Figure 2.1: Basic HTML Table

[8]http://getbootstrap.com/
[9]http://lesscss.org/
[10]http://sass-lang.com/libsass

Bootstrap offers flexibility and reusability, and that's the main reason why a lot of developers have been working into web site templates (free and commercial) at the top of the bootstrap functionality, adding their own extensions and producing an easy visual solution to the graphical user interface of a web application.

In the case of this project, we have explored some options and chosen AdminLTE[11].

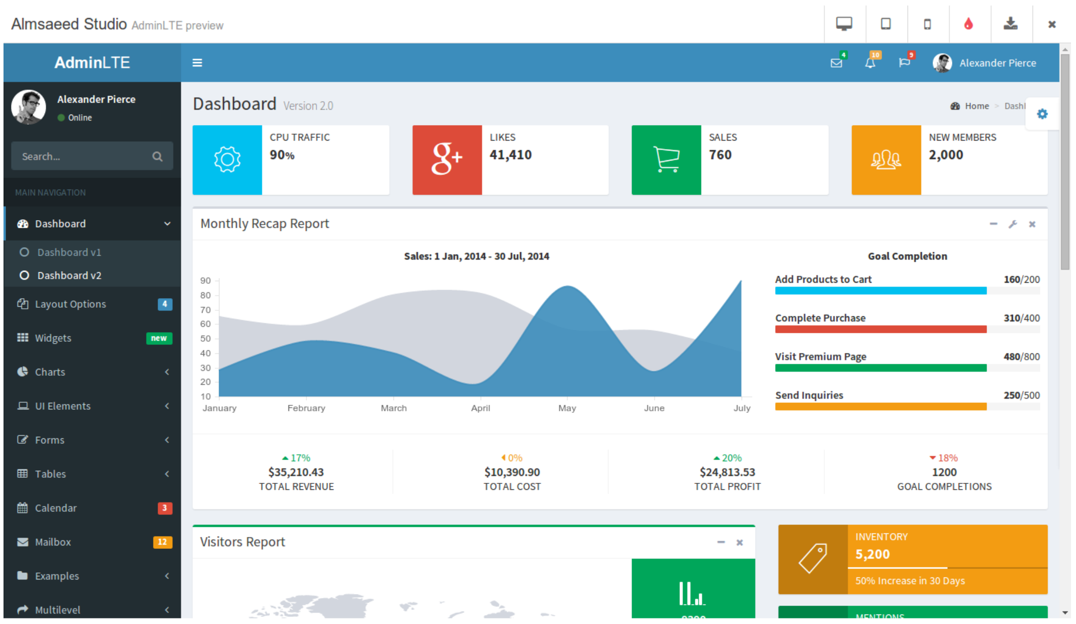


Figure 2.2: adminLTE template

[11]https://almsaeedstudio.com/

This bootstrap template is free and open source, built on top of bootstrap 3, responsive, easy to customize and has an active community. But further than the visual style, this template offer us a set of features very interesting for our purposes in the data analysis.

It implements libraries as Flot.js[12] and Morris.js[13], two graph renderers that we could eventually inject in our dashboards with some works transforming them into Sefarad widgets (See widgets chapter).

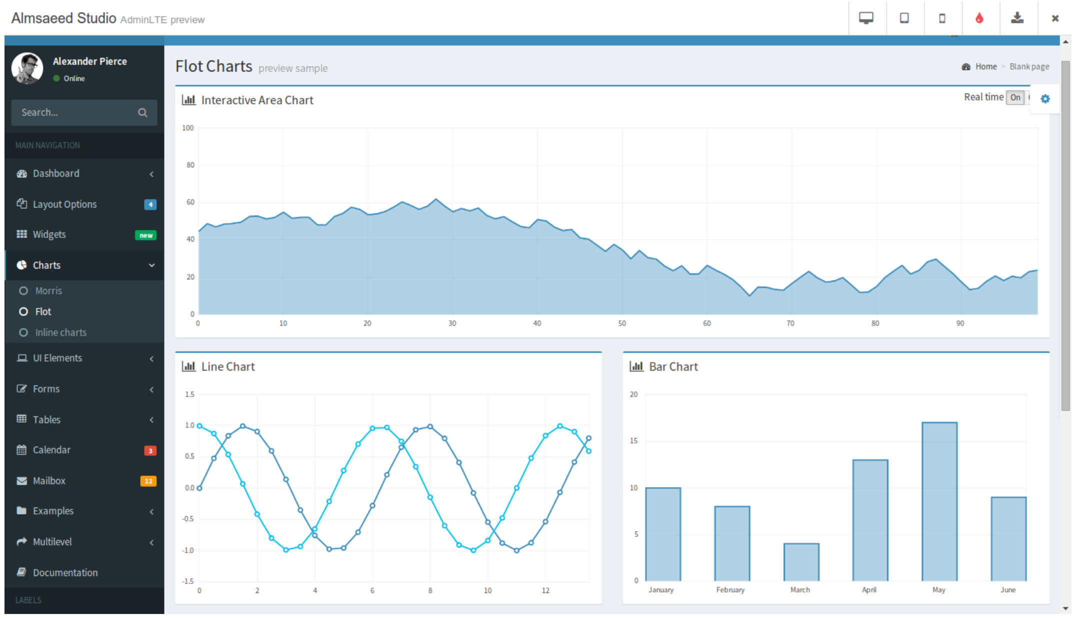


Figure 2.3: Premade Charts

It also provides some useful mechanisms as 404 error screen, log-in/log-out windows, lockscreens, etc.

It definitely provides a good graphical source for future development and it would be interesting to continue integrating its widgets as Sefarad's.

It presents some problems though at the time of using its functionality inside Polymer components. We will explore this problem in chapter 3.

[12]http://www.flotcharts.org/
[13]http://morrisjs.github.io/morris.js/

2.4.2 Leaflet

Leaflet[14] is a modern open-source JavaScript library for interactive maps. Leaflet is designed with simplicity, performance and usability in mind. It works efficiently across all major desktop and mobile platforms out of the box, taking advantage of HTML5 and CSS3 on modern browsers while still being accessible on older ones. It can be extended with a huge amount of plugins, has a beautiful, easy to use and well-documented API and a simple, readable source code. Weighing just about 33 KB of JS code, it has all the features most developers ever need for online maps.

It offers support to markers and pop-up binding, that we will use extensively in our demos.

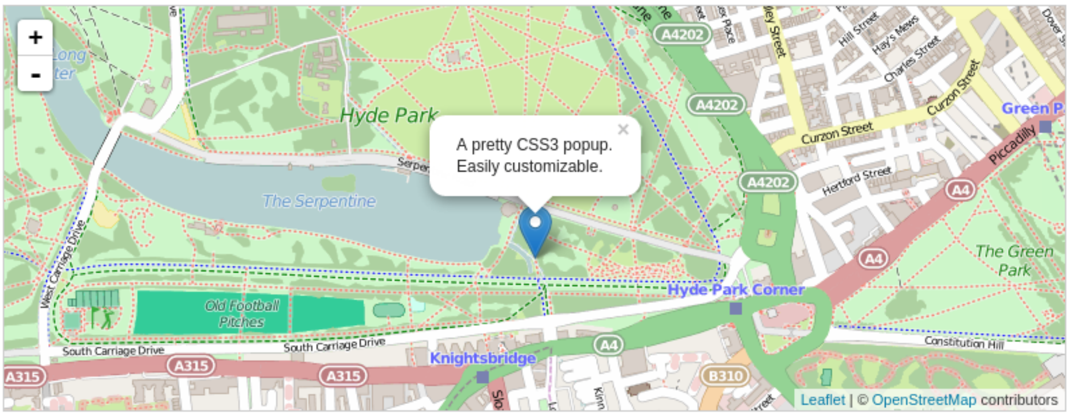


Figure 2.4: Leaflet Marker and pop-up

We chose this framework as a replacement of Open Layers 3 due to the features it provides for geoJSON polygon and multi-polygon rendering, with an automatic coordinate conversion between the base layer and the polygons definition. GeoJSON is becoming a very popular data format among many GIS technologies and services — it's simple, lightweight, straightforward, and Leaflet is quite good at handling it.

A GeoJSON object may represent a geometry, a feature, or a collection of features. GeoJSON supports the following geometry types: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeometryCollection. Features in GeoJSON contain a geometry object and additional properties, and a feature collection represents a list of features.

[14] http://leafletjs.com/

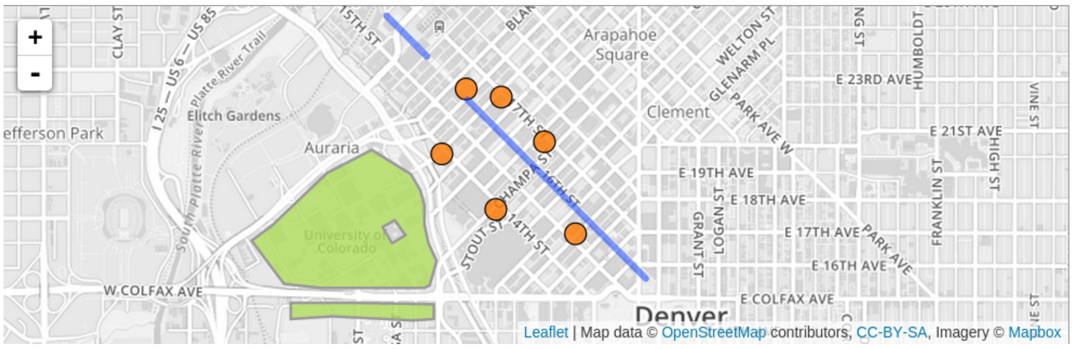


Figure 2.5: Leaflet GeoJSON Layer

In 2.4 we have listed an example of a simple GeoJSON feature.

Listing 2.4: geoJSON example

```
var geojsonFeature = {
    "type": "Feature",
    "properties": {
        "name": "Coors Field",
        "amenity": "Baseball Stadium",
        "popupContent": "This is where the Rockies play!"
    },
    "geometry": {
        "type": "Point",
        "coordinates": [-104.99404, 39.75621]
    }
};
```

Listing 2.4: geoJSON example

GeoJSON objects are added to the map through a GeoJSON layer. The style option can be used to style features, even based on their properties as we can observe in the code block 2.5.

```
L.geoJson(geojsonFeature).addTo(map);
L.geoJson(states, {
    style: function(feature) {
        switch (feature.properties.party) {
            case 'Republican': return {color: "#ff0000"};
            case 'Democrat':   return {color: "#0000ff"};
        }
    }
}).addTo(map);
```

Listing 2.5: Styling By Property Example

The onEachFeature option is a function that gets called on each feature before adding it to a GeoJSON layer. A common reason to use this option is to attach a popup to features when they are clicked, as we list in 2.6.

```
function onEachFeature(feature, layer) {
    // does this feature have a property named popupContent?
    if (feature.properties && feature.properties.popupContent) {
        layer.bindPopup(feature.properties.popupContent);
    }
}
```

Listing 2.6: Pop-up Binding

2.4.3 Crossfilter

Crossfilter[15] is a JavaScript library for exploring large multivariate datasets in the browser. It supports extremely fast (less than 30 milliseconds) interaction with coordinated views. Since most interactions only involve a single dimension, only small adjustments are made to the filter values, so incremental filtering is significantly faster than starting from scratch. Crossfilter uses sorted indexes to make this possible, dramatically increasing the performance of live histograms and top-K lists.

2.4.4 Dc.js

dc.js[16] is a javascript charting library with native crossfilter support that enables highly efficient exploration on large multi-dimensional dataset. It uses d3 engine to render charts in css friendly svg format. Charts rendered using dc.js are naturally data driven and reactive therefore providing instant feedback on user's interaction. dc.js provides us a great library of charts ready to use, as we illustrate in figure 2.6.

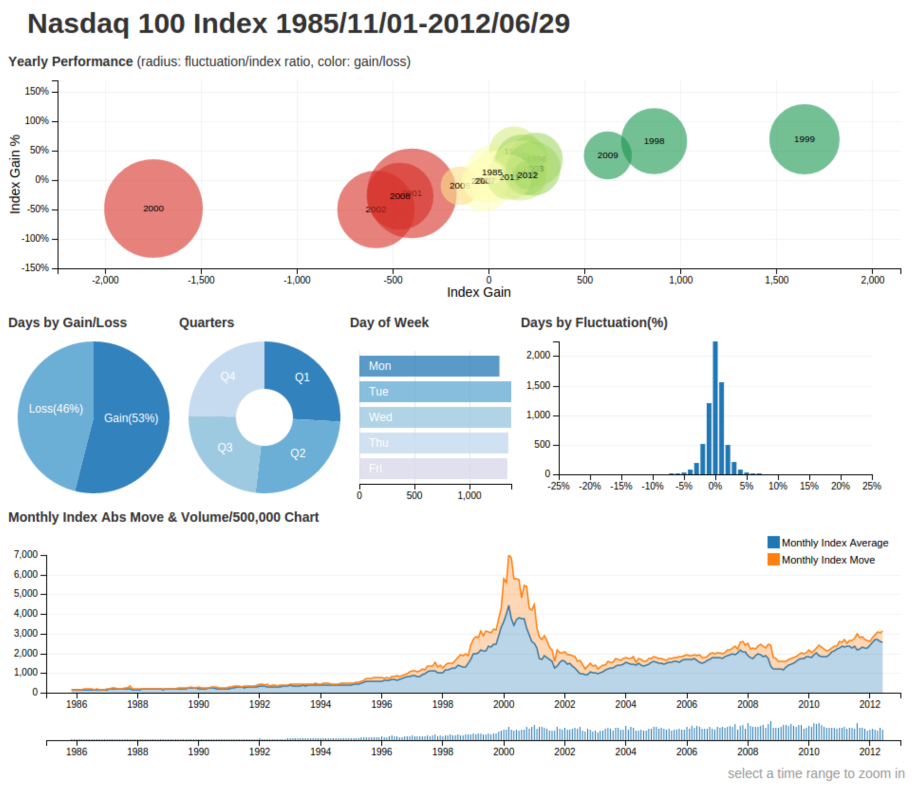


Figure 2.6: dc.js Charts

[15]http://square.github.io/crossfilter/
[16]http://dc-js.github.io/dc.js/

Besides, it offers a base chart that can be used to build a graphical interface on it. Base chart is an abstract functional object representing a basic dc chart object for all chart and widget implementations. Every function on the base chart are also inherited available on all all chart implementations extending base chart in dc library. We will use this one in a great number of widgets, and propose this template (after being integrated into Polymer) as the starting point at the task of constructing a custom widget for Sefarad-2.0.

Despite all these great capabilities, dc.js still has some problems, as the initialization of widgets at runtime. We will detail this problem in chapter 3.

2.5 Server-side web technologies

2.5.1 MongoDB

MongoDB[17] is an open-source document-oriented NoSQL database distributed under the GNU Affero General Public License[18] and the Apache License[19].

MongoDB is structured into collections instead of table-based relations. Those collections are a set of BSON (Binary JSON) documents containing a set of fields or key-value pairs: keys are string and values cans be of so many types (string, integer, float, timestamp, etc.).

In MongoDB, information is stored in form of documents which doesn't have a predefined schema. In place of that, these documents have a JSON array structure with multiple depth levels.

MongoDB is optimized for query 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 makes the system highly scalable.

This fits perfectly with our needs. We will use MongoDB to store user accounts, data-sources definitions, queries and their results in json or geo-json, output formats from SPARQL endpoints.

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

Moreover, we can use mongoDB as a datasource itself, and support queries. In block code 2.7 we show an example of a query pointed to a mongoDB.

```
{"$or": [{"year": {"$gt":1965,"$lt":2010}}, {"rating": {"$gt":6}}]}
```

Listing 2.7: Map Widget onClick Code

This query selects all movies made between 1965 and 2010 or with a minimun rate of 6, expecting a response in JSON format.

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 easy to communicate and manipulate the data. In the case of this project, we create a mongoDB and connect to it with the functions of the DART framework.

2.5.2 Fuseki

Fuseki [20] 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 for storing our own RDF files containing geoSPARQL data as a external tool for Sefarad datasets.

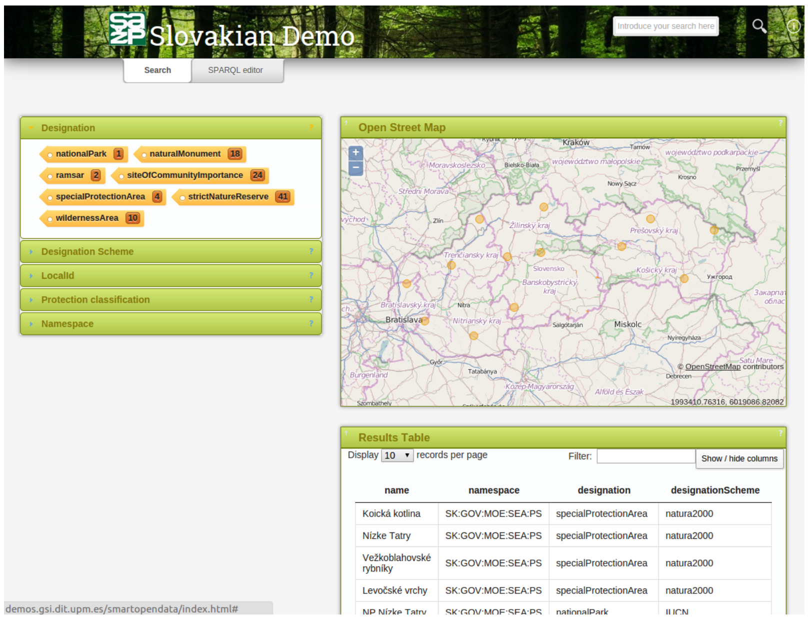
[20]https://jena.apache.org

2.6 Sefarad 1.0

Sefarad[13][14] 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. 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[21] and follows a Model View-View Model (MVVM) pattern performed with the Knockout[22] 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 SPARQL Editor. 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 SPARQL Editor tab we can write and preview queries to a defined endpoint.



Display 10 records per page Filter: Show / hide columns

name	namespace	designation	designationScheme
Koická kotlina	SK:GOV:MOE:SEA:PS	specialProtectionArea	natura2000
Nízke Tatry	SK:GOV:MOE:SEA:PS	specialProtectionArea	natura2000
Vežkoblahovské rybníky	SK:GOV:MOE:SEA:PS	specialProtectionArea	natura2000
Levočské vrchy	SK:GOV:MOE:SEA:PS	specialProtectionArea	natura2000
NP Nízke Tatry	SK:GOV:MOE:SEA:PS	nationalPark	IUCN

Figure 2.7: Sefarad 1.0

[21]http://www.w3.org/TR/html5/
[22]http://knockoutjs.com/

We will use this framework as a guide and reference to develop our own framework implementing the same functionality and growth capabilities.

2.7 Summary

We have seen in this chapter a couple of distinct technologies that we use in our final prototype:

- Linked Data: We use the concepts of linked data philosophy querying RDF datasets through SPARQL queries. On top of that basis we use geoSPARQL directives to take advantages of Geo Linked Data location features.
- Web Components: we make from web components our basic widget structure, taking into consideration its four principles.
- Polymer: It is the web components concrete framework that we have. It provides us an easy way of implementing web components in our web.
- Mongo DB: offers us a repository where we will store user data as data sources and queries definitions, as well as this kind of databases will serve us as data sources.
- Bootstrap - AdminLTE: We use bootstrap to style our web and take advantage of its style encapsulation. In this context, on top of bootstrap, we choose AdminLTE as a template to construct our graphical interface.
- Leaflet: We use Leaflet to implement a map widget with geoJSON compatibility and an interesting set of features.
- Dc.js - Crossfilter: We use Crossfilter as the filtering core of our framework and, on top of that, Dc.js makes charts for us and manages them tracking their filters and updating their graphics.
- Sefarad 1.0: It is our predecessor and, therefore, our reference in matter of features and interaction design.

CHAPTER 3

Sefarad 2.0

In this chapter we will describe the details of how Sefarad 2.0, our final prototype, works. We will present all distinct parts that constitute the application and the modules that form each one. We will define the interaction model that we have design for its use and then we will present each widget developed for the case studies and how to use it.

3.1 Introduction

Sefarad 2.0 has been under development for about a year, and has become a fairly complex piece of software that integrates some emerging technologies. This integration is not trivial and may need a further explanation for a new developer. This chapter is a detailed description of how Sefarad 2.0 works. We will start with the architecture, explaining all modules that take part in the program and how they relate to each other. Then we will move on defining the interaction model of Sefarad or how is it planned to be used. This section could be thought as a use manual. After that, we will list each Sefarad's widget and explained how is it constructed and how it can be used.

In chapter 5 we will follow the steps we took to develop Sefarad 2.0 from Sefarad 1.0, listing all prototypes we developed on the way and comparing them.

3.2 Architecture

Sefarad 2.0 is formed from various pieces or modules of software working together. At a wide sense, it can be divided into two main module groups: the dashboard modules and the queries pipeline modules. Both of them touch a mongoDB, containing the user defined data and being a nexus between the two groups.

The dashboard is mainly made of Web Components, that take data, render it, and provide user interaction; and a controller module that provides the necessary backup for the widgets to use Crossfilter and Dc.js. This controller will also feed the widgets at the initialization stage, communicate with the mongoDB and query the corresponding data source.

On the other hand, the queries pipeline modules are meant to define and modify data sources and queries and push them to the mongoDB repository, so the dashboard modules are able to retrieve and work with them. We have two modules here, each of them dedicated to one type of element, data source or query.

The architecture of the application can be resumed in the graph 3.1.

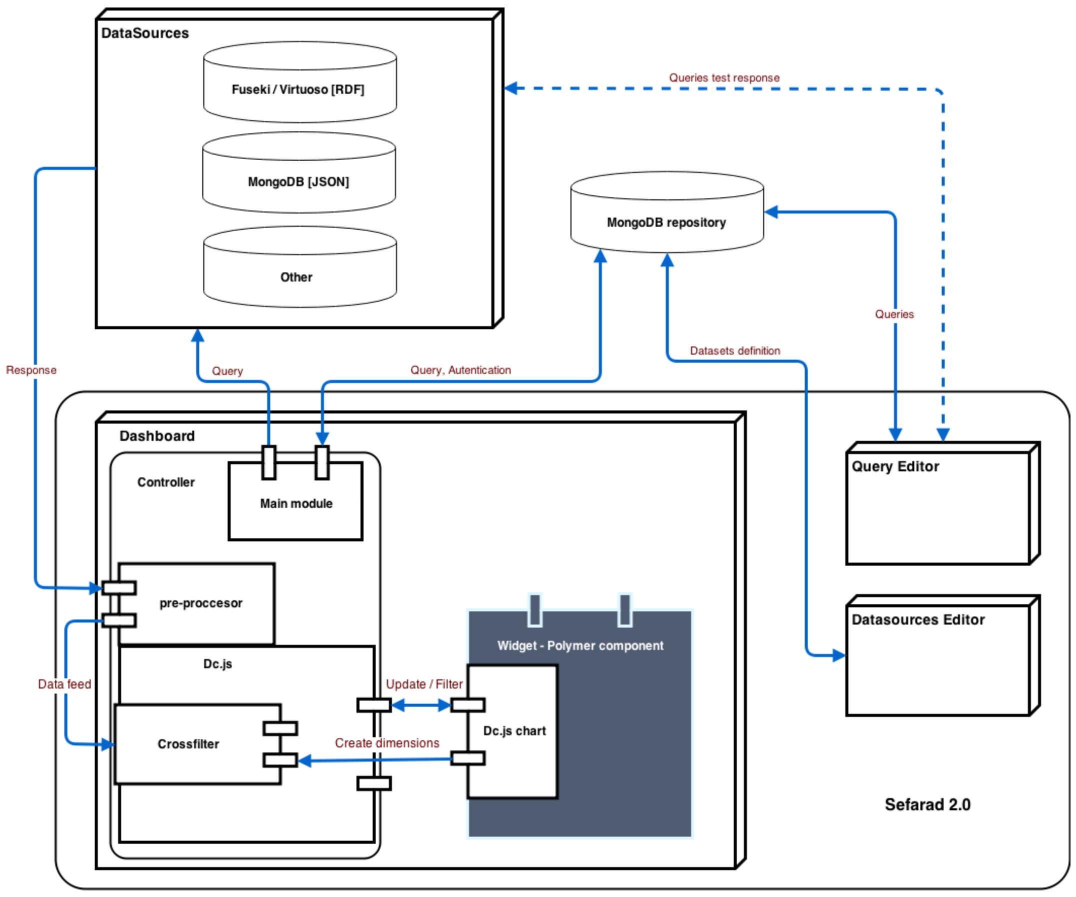


Figure 3.1: Sefarad 2.0 Architecture

This architecture has some advantages and disadvantages, mainly associated to the use of web components and Dc.js, new technologies not thought to be use as we do here.

Some of the advantages that Sefarad 2.0 exhibits are: The code is modular and easily expandable. Thanks to the use of the web component technology each module is small and readable, it's easy to focus on one part of the code and it favours team work as the influence between modules is minimized, so we can lose care about interfering with another team mate's work. Dc.js provides filter management and data spreading between our widgets, so we don't have to program or take care about these low level task. In this architecture we have separated the chart creation from the main object so the number of widgets is not relevant to Dc.js in order to work, it will be able to identify and track all widgets without problems.

Nervertheless, the use of components has brought some drawbacks, that are likely to be overcome once this technology reachs a certain degree of maturity. One example is the impossibility to add new charts at runtime. This is true due to the nature of Dc.js, that creates an unique element inside the controller module at the initialization time when it has already retrieve all widgets. Creating a new widget and reconstructing the Dc.js object would require necessarily to reload web page and reconstruct all widgets.

There is also a mayor disadvantage on the use of web components. Web components use shadow DOM, an interesting technology that, at counter part, impede us from using jQuery. This is an important issue, as there are a huge number of interesting jQuery plugins that we might be interested in, but that we will find problems at the time of integrating them. Although hard, it is not impossible to adapt a jQuery plugin into a web component. All we have to take care is wrapping all jQuery DOM objects retrieving with functions that respect the shadow DOM.

We provide an example of solving this problem with the leaflet map widget.

Lets break this architecture into pieces and explain further the function of each one.

3.2.1 Dashboard Architecture

The dashboard is driven by some modules that interact between them: the main module, the data pre-processor, web components and Dc.js-Crossfilter.

3.2.1.1 Main module

The dashboard is controlled from a main module. This module is written in javascript and Dart but, in the future, it will be translated to pure Dart.

This main module has two functions, query a data source and authenticate users.

To query a data source, the module must access to our mongoDB repository to retrieve which query we have to execute and which endpoint we have to point to. Currently this feature is under development so, instead of retrieving that information from mongoDB, it is contained in the HTML document specific of the dashboard.

Once the query and data source are defined, the main module executes the query with the language-specific code. The example listed below (3.1) illustrates how to fire SPARQL queries:

```
var getPolygonsFromEuro = function () {

    if (currentQuery != lastQuery) {
        var polygonsfeuro_query = currentQuery;
        var temporal = queryEndPoint + '?query=' +
            encodeURIComponent(polygonsfeuro_query);
        var req = new XMLHttpRequest();
        req.open("GET", temporal, true);
        var params = encodeURIComponent(polygonsfeuro_query);
        req.setRequestHeader("Content-type", "application/x-www-
            form-urlencoded");
        req.setRequestHeader("Accept", "application/sparql-results+
            json");
        req.send();
        console.log("polygons query sent");
        req.onreadystatechange = function () {<polymer-element name
            ='generic-widget' class='dc-element' attributes='param'>
            if (req.readyState == 4) {
                if (req.status == 200) {
                    console.log("polygons query response received")
```

```
                                   ;
                              var res = eval("(" + req.responseText + ")");
                              var data = JSON.stringify(res.results.bindings)
                                   ;
                              rawData = JSON.parse(data);
                              dataReady = true;
                         } else {
                         }
                    }
               };
               return false;
          }else
          {
               $.getJSON("assets/cache.json", function(result){
                    console.log("polygons query response picked from local"
                         );
                    var data = JSON.stringify(result);
                    rawData = JSON.parse(data).results.bindings;
                    dataReady = true;
               });
          }
};
```

Listing 3.1: Execute Query Code

Regarding authentication and authorization, the design is based on Dart facilities backed by mongoDB (3.2). It checks the mongoDB repository at starts a new session in case of success:

```

class SignGoogle {
  static bool logged = false;
  static var host = "localhost:1990";

  void login() {
    InputElement ie = querySelector("#password");
    if (ie.value == "****") googleLogin.login();
    else {
      ie.value = "";
      window.alert("Invalid Admin Password");
    }
  }
  String getHost() => host;
  bool isLogged() => logged;

  void logout() {
    googleLogin.logout();
    logged = false;
    resetProfile();
  }

  final googleLogin = new GoogleOAuth2(
      "675126827387-jrvg34sf52dni2o8o5kjgthc3abm8s0u.apps.
          googleusercontent.com",
      ["openid", "email"],
      tokenLoaded:loginCallback);

  static void loginCallback(Token clave) {
    final googlePlusURL = "https://www.googleapis.com/plus/v1/
        people/me";
    var request = new HttpRequest();
    request.open("GET", googlePlusURL);
    request.setRequestHeader("Authorization", "${clave.type} ${
        clave.data}");
    request.onReadyStateChange.listen((_) {
      if (request.readyState == HttpRequest.DONE &&
      (request.status == 200 || request.status == 0)) {
        printProfile(request.responseText);
        logged = true;
      }
    });
    request.send();
  }
[...]
}
```

Listing 3.2: Authentication Code

3.2.1.2 Data pre-processor module

This module takes the data retrieved from the query and is thought as a space for the application programmer for changing all parameters needed, so Crossfilter will receive later proper values. This can be used to fix bad values in certain parameters, or for unifying values that are not expressed the same way. Below (3.3) we can see an example of one of this blocks, designed for the restaurants demo:

```
rawData.forEach(function(d) {

  //Unifying reservations facet values
    var s = d.reservations.value.trim();
    var sub = s.substr(s.length - 3);
    d.newReservations = {};
    if (sub == 'Yes') {
        d.newReservations.value = 'yes';
    }
    else {
        d.newReservations.value = 'no';
    }

  //extract the name from the element URI
    var st = d.d.value;
    d.name = {};
    d.name.value = st.substr(st.lastIndexOf('/') + 1, st.length).
        replace(/-/g, " ");

  //Give each element an unique ID
    rawData.forEach(function(d) {
        d.total = 1;
        d.id = idGen;
        idGen++;
    });
});
```

Listing 3.3: Pre-processor Code

With this code we fixed the retrieved data, so it is ready now for Crossfilter to process.

3.2.1.3 Web components

Web components are a main feature of Sefarad 2.0, and in this section we will show how we exploit their capabilities.

All widgets in Sefarad 2.0 are web Components (used under the Polymer implementation) and, as them, they are ideally encapsulated and isolated from the rest of the code.

First, we have a special zone 3.4 in the HTML of each dashboard to import all widgets' web Components.

```
<!-- POLYMER COMPONENTS IMPORT ZONE -->

    <link rel="import" href="polymer-elements/pie-chart/pie-chart.
        html">
    <link rel="import" href="polymer-elements/bar-chart/bar-chart.
        html">
    <link rel="import" href="polymer-elements/leaflet-map/leaflet-
        map.html">

<!-- end/ POLYMER COMPONENTS IMPORT ZONE -->
```

Listing 3.4: Web Components Import

Then, each widget is instantiated in the dashboard through its custom HTML tag. For example, if we have imported a faceted-search widget and now we want to include it in our dashboard, the required code would be the one listed below (3.5).

```
<!-- faceted-search -->
<!--Widgets must have 'widget' class in order to render their
    loading screen-->
< div class = "box box-primary widget" >
    < div class = "box-header" >
    < i class = "glyphicon glyphicon-search" > < /i> < h3 class = "
        box-title" > Search < /h3> < /div>
    <!-- box-body -->
    < div class = "box-body chart-responsive" >

        <!-- WEB COMPONENT -->
        < faceted-search params = "designation, designationScheme"
            class = "dc-element" >
        < /faceted-search>
        <!-- /.WEB COMPONENT -->


    < /div>
    <!-- /.box-body -->
    < /div>
<!-- /.faceted-search -->
```

Listing 3.5: Web Components Insertion

Reading carefully, it can be noticed that the entire widget is encapsulated inside the faceted-search tag, with its parameters set as tag parameters, while the rest of the code is the box styled widget of AdminLTE.

A Polymer element is a separated HTML document. It has two main and different parts: An HTML template and a script.

The HTML template of the web component is the code that will be injected into our dashboard once the widget is initialized. Here we can write all the HTML code that we want to be rendered inside the widget. We can also use and import custom styles, that will not be affected then by the outside's style. The same issue occurs with the script of web components, it only affects to what is inside the template and doesn't reach the other widgets templates.

This code, therefore, is completely independent of the rest of the dashboard, and that is the main advantage that we receive from the use of web components.

So, a basic web component will have the form of the code listed in 3.6.

```
<link rel="import"
      href="../bower_components/polymer/polymer.html">

<polymer-element name="proto-element">

  <template>
    <span>Im <b>proto-element</b>. Web component prototype.</span>
  </template>

  <script>
    Polymer({
      ready: function() {
        //...
      }
    });
  </script>

</polymer-element>
```

Listing 3.6: Web Component Structure

Inside this HTML template we can use a number of useful features as data-bindings with the data model or auto iterations through data. We have an example of how to use this bindings in the code listing 3.7.

```
<polymer-element name="fav-color">

  <template>

    This is <b>{{owner}}</b> fav-color element.

    {{owner}} likes the color
    <span style="color: {{color}}">{{color}}</span>.

  </template>

  <script>
    Polymer({
      owner: "Daniel",
      color: "red"
    });
  </script>

</polymer-element>
```

Listing 3.7: Web Component Structure

In 3.7 we can observe that we bind a set of parameters from the web component's script with the template, being that relation completely isolated from the rest of the web's code.

We will go into detail of how we define our web components in section 4.3.1, "Widgets common implementation".

3.2.1.4 Crossfilter-Dc.js

Crossfilter and Dc.js are distributed between the main module and the widgets on the dashboard system.

At one hand, when the query data is retrieved and after the pre-processing module, we create the Crossfilter object. This Crossfilter object and the rawData (for render purposes) is then propagated to all widgets as shown in the 3.8 listing.

```
ndx = crossfilter(rawData);

var dcElements = $(".dc-element");
for (var i = 0; i < dcElements.length; i++) {

  //Crossfilter object Propagation
  dcElements[i].crossfilter = ndx;
  dcElements[i].geoJSON = rawData;

  //Widget initialization, where dimensions are created:
  dcElements[i].init();
}
dc.renderAll();
```

Listing 3.8: Crossfilter Initialization

On the other hand, after the injection, every widget gets its init() function called, as shown in the previous figure. In this function, each widget creates a new Crossfilter dimension and group and initializes a new Dc.js chart with them.

Here is the code for a pie chart as an example, please note the dimension and group definition over the Crossfilter object that we propagated in the controller code.

We also create the Dc.js charts here. With this, we are telling the Dc.js unique object, located at the controller level, that it needs to keep track of this chart and record its filter and update actions (3.9).

```
var p = this.param; //set the params from the HTML tag

//Dimension creation
this.dimension = this.crossfilter.dimension(function(d) {
    return d[p].value;
});

//Group creation
this.group = this.dimension.group().reduceCount();

//Dc chart creation
this.chart = dc.pieChart("#chart");
this.chart
    .ordinalColors(["#1E7751","#07A360","#21553F","#25342E"])
    .width(175).height(175)
    .dimension(this.dimension)
    .group(this.group)
    .innerRadius(30)
    .root()[0][0] = this.$.chart;
this.$.chart.classList.add("dc-chart");
```

Listing 3.9: Dc Initialization

In the Crossfilter initialization figure we call the dc.renderAll() function at the end of the code. When all charts are set (after the init() function of each web component) we will render them all at once, so there will be an unique Dc.js object maintaining a list of all active filters and managing all Crossfilter dimensions.

Now, a few words about Crossfilter dimensions: Dimensions are bound to the Crossfilter once created. Creating more than 8 dimensions, and more than 16 dimensions, introduces additional overhead. More than 32 dimensions at once is not currently supported, but dimensions may be disposed of using "dimension.dispose()" function to make room for new dimensions. Dimensions are stateful, recording the associated dimension-specific filters, if any. Initially, no filters are applied to the dimension: all records are selected. Since creating dimensions is expensive, you should be careful to keep a reference to whatever dimensions you create.

3.2.2 Queries pipeline Architecture

As a separate function of Sefarad 2.0, we have the possibility of defining our own data sources as named URI endpoints, in order to define later our queries to those endpoints. Having our queries we can modify them and test them against their endpoint, previewing the results.

We will present the complete Sefarad workflow in the "Interaction model" section. For now, we will focus in how it works:

Sefarad has his own Dart server, from which we listen POST and GET petitions to register and retrieve data. This server connects to a mongoDB database to store all data. Below (3.10) we have the code that creates the mongoDB database and handles the http GET requests:

```
/**
 * Handle GET requests by reading the contents of data.json
 * and returning it to the client
 */
void handleGet(HttpRequest req) {
    HttpResponse res = req.response;
    RegExp regex = new RegExp("([^?=&]+)(=([^&]*))?");
    print("${req.method}: ${req.uri.path}")
    String data_file = path + req.uri.path.substring(1, req.uri.
        path.length) + ".json";
    addCorsHeaders(res);

    if (req.uri.path == "/mongoDBquery") {
        [...]
        Db db = new Db(database);
        DbCollection coll;
        ObjectId id;
        db.open().then((c) {
            print('connection open');
            coll = db.collection(collection);
            res.headers.add(HttpHeaders.CONTENT_TYPE, "application/
                json");
            try {
                print(JSON.decode(queryStr));
                Cursor cursor = coll.find(JSON.decode(queryStr));
                cursor.forEach((Map v) {
                    var id = v["_id"];
                    v.remove("_id");
                    v["_id"] = id.toString();
                    mongoDB.add(v);
                }).then((dummy) {
                    res.statusCode = HttpStatus.OK;

                    //Here we return the results
                    res.write(JSON.encode(mongoDB));
                    res.close();
                });
            }
            [...]
```

Listing 3.10: Dart Server Code

And here (3.11) we have the Dart code in charge of executing POST petitions and write data into mongoDB:

```

/**
 * Handle POST requests by overwriting the contents of data.json
 * Return the same set of data back to the client.
 */
void handlePost(HttpRequest req) {
    HttpResponse res = req.response;
    print("${req.method}: ${req.uri.path}");
    String data_file = path + req.uri.path.substring(1, req.uri.
        path.length) + ".json";
    BytesBuilder builder = new BytesBuilder();
    addCorsHeaders(res);

    req.listen((List < int > buffer) {
            builder.add(buffer);
            if (req.uri.path == "/login") {
                var data = new String.fromCharCodes(buffer);
                print(data);
                res.close();
            }
            var file = new File(data_file);
            var ioSink = file.openWrite(); // save the data to the
                file
            ioSink.add(buffer);
            ioSink.close();
            // return the same results back to the client
            res.add(buffer);
            res.close();
        },
        onError: printError);
}
```

Listing 3.11: Dart Server Code 2

As we can see, in the POST handler section we don't make distinctions between what type of data (queries or data sources) we are posting. This will be done from the specific code of each form, maintaining the code unique for both branches.

Now lets talk about how the server handles data sources and queries data:

We have an HTML document dedicated to the definition of datasources, which contains a form that pushes data into the server when the user defines a new datasource. Next (3.12), we have the JSON of a defined datasource.

```
{
    "Name": "Dbpedia",
    "Type": "Sparql",
    "Endpoint": "http://dbpedia.org/sparql?default-graph-uri=http
        ://dbpedia.org&query=",
    "Collection": "",
    "User": "",
    "Password": ""
}
```

Listing 3.12: Datasource definition JSON

We then POST these data to the server from the Dart code associated to these data sources definition page (3.13).

```
var queryVar = {
    "Name": name,
    "Type": type,
    "Endpoint": endPoint,
    "Collection": collection,
    "User": user,
    "Password": password
};
datasets.add(queryVar);
querySelector('#saveSuccess').classes.remove("hide");
String jsonData = JSON.encode(datasets);
var request = new HttpRequest();
request.onReadyStateChange.listen((_) {
    if (request.readyState == HttpRequest.DONE &&
        (request.status == 200 || request.status == 0)) {
        // data saved OK.
        print(" Data saved successfully");
    }
});
var url = "http://$host/web/dataset";
request.open("POST", url);
request.send(jsonData);
```

Listing 3.13: Datasources Data Sender

In the same way, we have a query definition page, where we defined through a form a query data JSON object. Our query definitions must be able to handle query parameters as we explain later. Here (3.14) we have an example of a query object in JSON format, as we will save and get in the mongoDB repository.

```
{
    "Name": "Nobel laureates in <fieldOfScience> ordered by <order>
         of birth date",
    "Query": "PREFIX : <http://dbpedia.org/resource/> [...]",
    "Type": "Sparql",
    "Parameters0": ["Nobel_laureates_in_Physics", "
        Nobel_laureates_in_Literature", "
        Nobel_laureates_in_Chemistry"],
    "Parameters1": ["ASC", "DESC"],
    "Parameters2": [],
    "Parameters3": [],
    "Parameters4": [],
    "Results": "[{\"nobel\":{\"type\":\"uri\ [...]"
}
```

Listing 3.14: Query definition JSON

This is a more complex data structure. We give the query a name, a type that we will use to edit it with an specific code editor, the parameters with the possible values and the result (later we can use this result in the dashboard instead of querying it directly).

The POST code, located in the Dart document associated with the query creator and editor form is listed below (3.15).

```
var queryVar = {
    "Name": name,
    "Query": dataQuery,
    "Endpoint": endPoint,
    "Type": type,
    "Parameters0": parameters0,
    "Parameters1": parameters1,
    "Parameters2": parameters2,
    "Parameters3": parameters3,
    "Parameters4": parameters4,
    "Results": ""
};
querys.add(queryVar);
querySelector('#saveSuccess').classes.remove("hide");
String jsonData = JSON.encode(querys);

var request = new HttpRequest();
request.onReadyStateChange.listen((_) {
    if (request.readyState == HttpRequest.DONE && (request.status
       == 200 || request.status == 0)) {
        print(" Data saved successfully");
    }
});
var url = "http://$host/web/queries";
request.open("POST", url);
request.send(jsonData);
```

Listing 3.15: Query Data Sender

At last, we will review the SPARQL editor itself. It is an integration of the YASGUI project[1] inspired by the demo developed within the EuroSentiment[2] project.

[1]http://yasgui.org/
[2]http://eurosentiment.eu/

When we write a query for Sefarad and want to include parameters with different values we include them in the query between 'minus', 'mayor'. This way, when the user selects a new value, our code loops through the text replacing these key words by their real value. This process of replacing is coded in the next code block (3.16).

```
function populateQueryWithParams() {
    [...]

    //Here is where we replace the parameters with the actual
        values,
    //using the events handler of the selectors:

    $("#selector" + k).change(function() {
        var regex2 = /selector(\S+)/i;
        var index = this.id.match(regex2);
        var selectedParameters = this.value;
        query = query.replace(match[index[1]], selectedParameters);
        match[index[1]] = selectedParameters;
        if (parsedData[j]["Type"] == "Sparql")
            yasqe.setValue(query);
        else
            $("#queryMongo").val(query);
    });

    //We then reload YASQUE to show the new filled query:
    if (parsedData[j]["Type"] == "Sparql") {
        yasqe.setValue(query);
    }
    else {
        $("#queryMongo").val(query);
    }
    return;
    [...]
}
```

Listing 3.16: SPARQL Editor Code

3.3 Interaction model

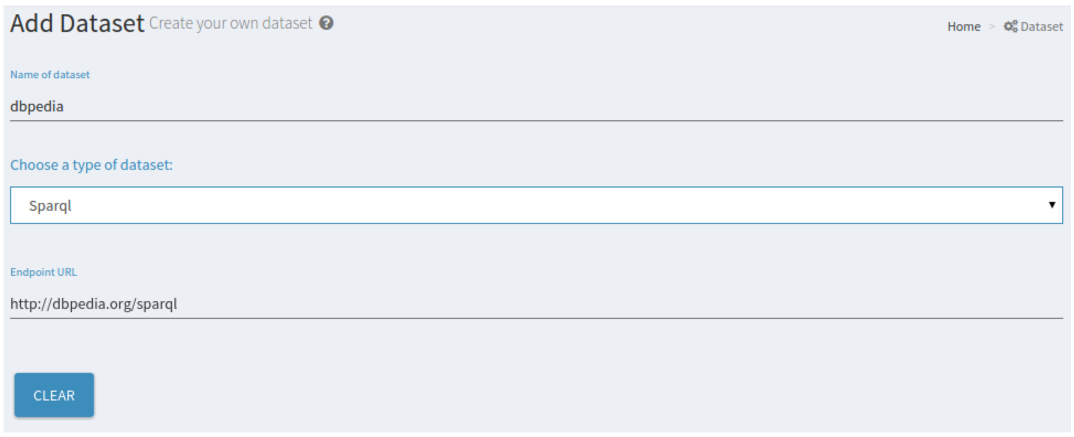
In this section we will describe the usability design of Sefarad, or how it is though to be used.

We will start by defining a data source through the user interface, and then we will see how to define a query with various parameters, watching its results in a prepared dashboard.

3.3.1 Defining a data source

First of all we need to define the endpoint where we will point our queries to. This source can be a SPARQL endpoint, as the fuseki server we are using for our Smart Open Data project (see use cases), or mongoDB databases containing our data.

We will be prompted with this screen:



Add Dataset Create your own dataset

Home > Dataset

Name of dataset: dbpedia

Choose a type of dataset: Sparql

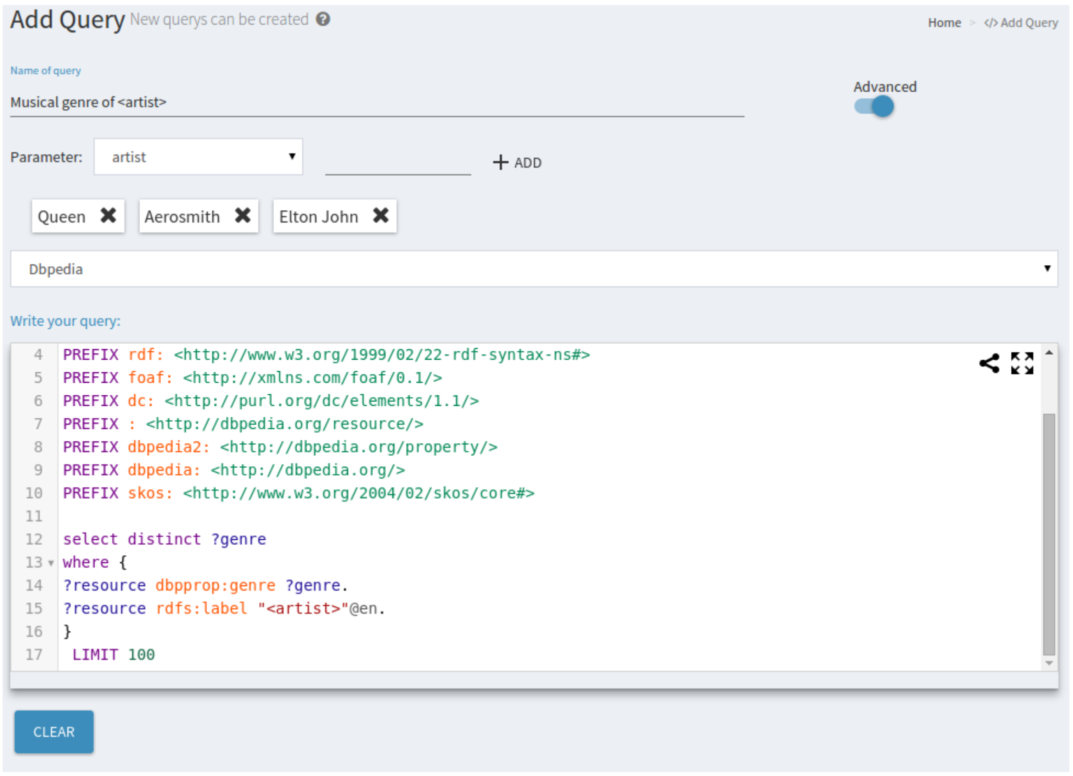
Endpoint URL: http://dbpedia.org/sparql

Figure 3.2: Add Data Source Web Page

Here we have to name the new data source, provide its valid URL and specify its type. The type specification is important because we give auto-completion and code correction support to various types of code, so we need to know the proper type to process it accordingly.

After these few steps we can click on the save button and our datasource will be ready to query from the rest of the elements of Sefarad.

3.3.2 Writing a query



Musical genre of <artist>

Parameter: artist

Queen, Aerosmith, Elton John

Dbpedia

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbpedia2: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>

select distinct ?genre
where {
?resource dbpprop:genre ?genre.
?resource rdfs:label "<artist>"@en.
}
 LIMIT 100
```

Figure 3.3: Add Query Web Page

This is the main query definition screen, the place where we will initially construct our query and its parametric structure (if desired that way).

As we can see, we can set various elements to configure our query. The objective of this page is to enable a non technical user to use this query without the need of using code, so we will have to do this coding part for him with an extra effort defining some parameters.

We have an "advanced mode" button at the right top of the page. Pressing this button we will enable the parameters option.

First of all, the query must have a name. In this name field is where we have to define the parameters of the query. These parameters are nouns that can be changed for distinct values in the query. For example, if we have a query that asks for the musical genre of a band or artist, this artist could be a parameter: *"Musical genre of <artist>"*.

Then, this "artist" parameter will appear in the "Parameters" section, and we will be able to add new values as "Elton John" or "Aerosmith".

Below these inputs we have to set the data source where we point our query.

Now its time to properly write the query in the query in the matching editor, that will be adequate for the type of data source selected. We just have to remember to write the parameters between the $<$ and $>$ math symbols, so the code an properly replace them later. Click the save button and all will be done.

3.3.3 Visualising and editing a query

Each dashboard that we create comes associated to a particular query. This query will be automatically executed when the dashboard is loaded and its results are rendered in the existing widgets.

But we have the choice to view and edit this associated query. In each dashboard we have the option to display the query editor, with a tab control at the top of the page:

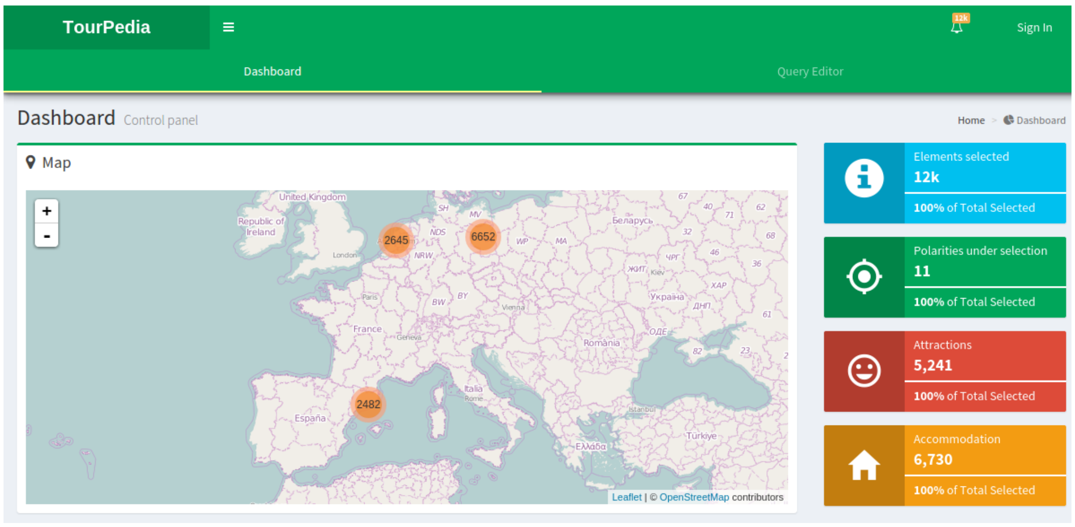
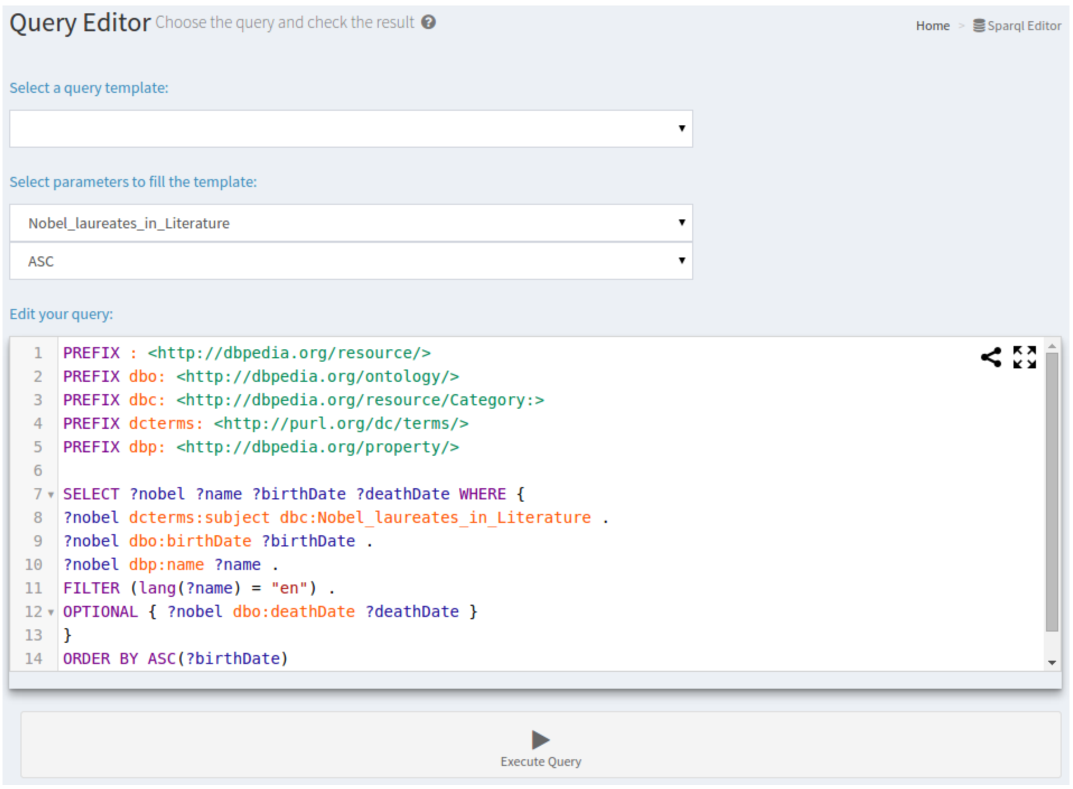

TourPedia
Sign In
Dashboard
Query Editor
Dashboard Control panel
Home > Dashboard
Map
Elements selected 12k 100% of Total Selected
Polarities under selection 11 100% of Total Selected
Attractions 5,241 100% of Total Selected
Accommodation 6,730 100% of Total Selected
Leaflet | © OpenStreetMap contributors

Figure 3.4: Dashboard Tabs

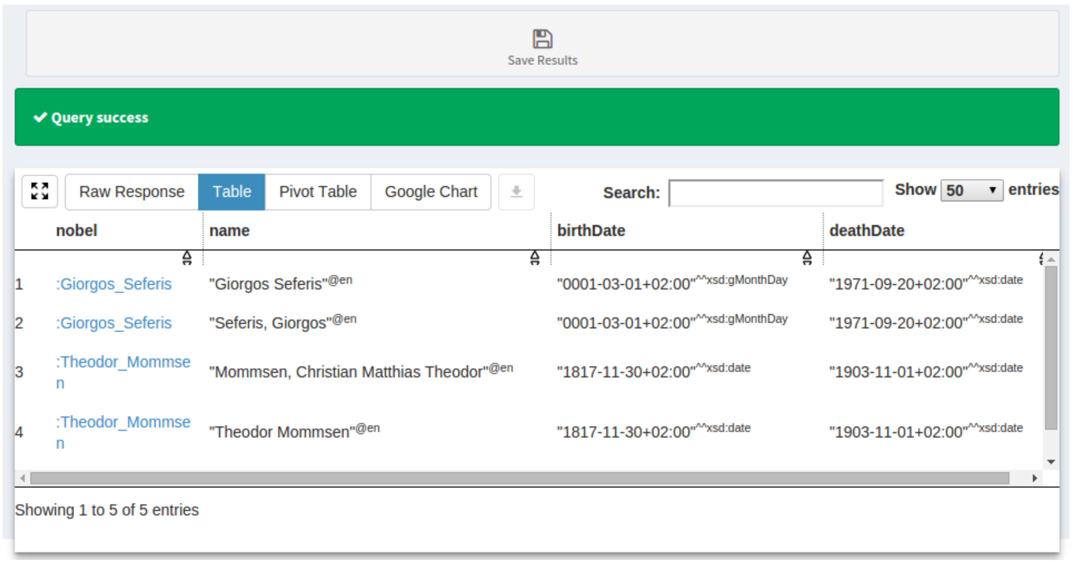
So, if we choose to see the query editor we will be prompted with this interface:

Here we can see the query's code and can modify the value of the query parameters. When we change the parameters on the selector, the code will be automatically changed, so there is no need to know the query language to change it.


Query Editor Choose the query and check the result
Home > Sparql Editor
Select a query template:
Select parameters to fill the template:
Nobel_laureates_in_Literature
ASC
Edit your query:
Execute Query

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbc: <http://dbpedia.org/resource/Category:>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX dbp: <http://dbpedia.org/property/>

SELECT ?nobel ?name ?birthDate ?deathDate WHERE {
?nobel dcterms:subject dbc:Nobel_laureates_in_Literature .
?nobel dbo:birthDate ?birthDate .
?nobel dbp:name ?name .
FILTER (lang(?name) = "en") .
OPTIONAL { ?nobel dbo:deathDate ?deathDate }
}
ORDER BY ASC(?birthDate)
```

Figure 3.5: Query Editor



	nobel	name	birthDate	deathDate
1	:Giorgos_Seferis	"Giorgos Seferis"@en	"0001-03-01+02:00"^^xsd:gMonthDay	"1971-09-20+02:00"^^xsd:date
2	:Giorgos_Seferis	"Seferis, Giorgos"@en	"0001-03-01+02:00"^^xsd:gMonthDay	"1971-09-20+02:00"^^xsd:date
3	:Theodor_Mommsen	"Mommsen, Christian Matthias Theodor"@en	"1817-11-30+02:00"^^xsd:date	"1903-11-01+02:00"^^xsd:date
4	:Theodor_Mommsen	"Theodor Mommsen"@en	"1817-11-30+02:00"^^xsd:date	"1903-11-01+02:00"^^xsd:date

Showing 1 to 5 of 5 entries

Figure 3.6: Query Editor

After modifying the query you can execute it with the button below the query's text. When this happens, the query results will be rendered in a YASGUI element. We have various ways of viewing the information: the raw response, on a table, on a pivot table or a google chart.

Once we are fine with the results we have the option of saving the query (with the so named button) so the next time we visit the dashboard it will show those results.

3.4 Widgets

In this section we will describe the widget system implemented in Sefarad-2.0. The first sub-section will be an in-depth description of the common structure of every widget in Sefarad and how they work. The second sub-section will be committed to a detailed description of each widget we have developed until now.

3.4.1 Common implementation

Each widget in Sefarad must implement a common structure to work correctly for the system. The outer shell of this structure is a Polymer element.

This Polymer element offers us a HTML template that each widget will fill differently, a place for concrete styling, and an interface with the HTML of the dashboard and the rest of the script code.

Lets focus now on the the JavaScript section of a Polymer web component. All widgets contain these properties there:

- Crossfilter Object
- Dc.js Chart
- Crossfilter Dimension
- Crossfilter Group
- HTML parameters (optional)
- ”Ready()” function
- ”Init()” function

The main Sefarad-2.0 script will wait until the query data is retrieved and parsed on one hand, and until all widgets are ready on the other. For that sake, every widget in Sefarad will have to implement the css class ”dc-element”. The main script will count how many widgets are in the dashboard on the page load event and will save that count number. From then on, it will check periodically if the data is ready and if that number has reached zero.

But, how will it reach zero? Every widget will implement a ”Ready()” function, that will access to that counter and reduce it in one unit. With this trick we can make sure that all widgets are ready at the moment of initializing them with the data. This is a problem derived of Polymer's operation. Polymer works in parallel with our main script, so we could experience an scenario where our code would be ready to execute but Polymer was still loading templates and assets.

Once all widgets are ready and the data is parsed, the code will create a new Crossfilter object over the data and iterate through every widget. In this loop, we are setting its Crossfilter Object property with this new one and calling its "Init()" function. That function will first initialize the widget parameters with those passed through the custom HTML tag. Secondly, it will create the Crossfilter dimension form the Crossfilter object that we injected from the outside and with the name of the data field we want to filter. At last, we will do the specific rendering and construction work that, in case of Dc.js charts, will be instantiating the chart and storing it in the corresponding widget property.

An empty and generic widget in Sefarad would look like the following code (3.17).

```
<template>
     <!-- HTML content --> <!-- css style -->
</template>
  <script>
       Polymer({
            crossfilter: {},
            chart: {},
            dimension: {},
            group: {},
            ready: function(){
                 numWidgets = numWidgets - 1;
            },
            init: function (){
                 var p = this.param; //params from the HTML tag
                 this.dimension = this.crossfilter.dimension(
                     function(d) {
                      return d[p].value;
                 });
                 this.group = this.dimension.group().reduceCount();
            }
       });
  </script>
```

Listing 3.17: Generic Widget Structure

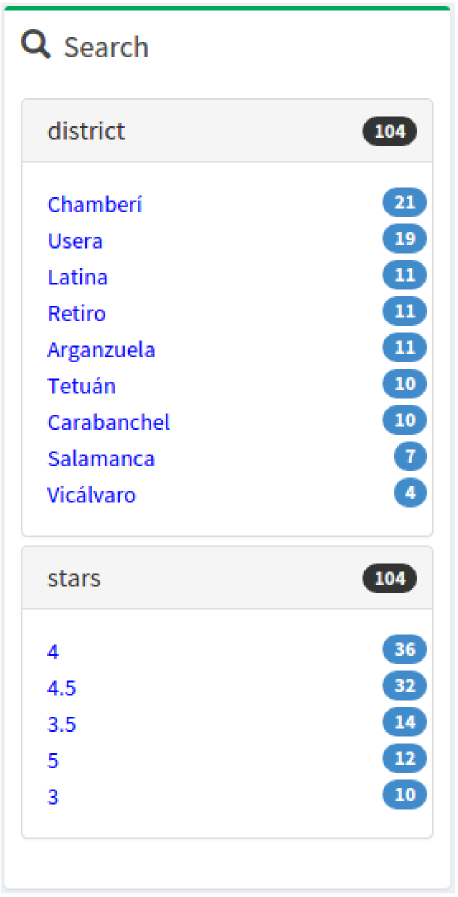
3.4.2 Specific implementation

In this section we will explain the implementation details of each widget of Sefarad 2.0, presenting the initial problem to solve, the solution and the problems or limitations of each technology used.

3.4.2.1 Faceted Search

We needed a faceted search widget like the one developed for the previous version of Sefarad. This widget must integrate with the rest of the widgets, so it must implement at least one Dc.js chart that makes it able to filter data.

We decide to develop a brand new widget with an accordion layout and based on bootstrap style. The widget will have one section for each parameter of the data. Inside this sections the different values with their counters will be displayed. These values will be interactive so if we click on them, they will toggle filters.



district	104
Chamberí	21
Usera	19
Latina	11
Retiro	11
Arganzuela	11
Tetuán	10
Carabanchel	10
Salamanca	7
Vicálvaro	4

stars	104
4	36
4.5	32
3.5	14
5	12
3	10

Figure 3.7: Faceted Search Widget

This code is developed under the template section of the web component. The HTML code used for this purpose is listed in 3.18. It uses two templates repeated iteratively inside an outer template div.

```
<template>    [...]
    <div id="accordion" class="panel-group">
      <template repeat="{{d in dimensions}}">
         <div class="panel panel-default">
            <div class="panel-heading">
               <h4 class="panel-title">
                 <div objectToHide="{{d[0]}}" on-click="{{
                     headerClick}}">{{d[0]}}
                   <span class="badge">{{d[4]}}</span>
                 </div>
               </h4>
             </div>
             <div id="{{d[0]}}" class="panel-collapse collapse in">
                <div class="panel-body">
                  <ul class="nav nav-pills nav-stacked">
                     <template repeat="{{p in d[2]}}">
                       <li dimension="{{d[0]}}" on-click="{{
                           valueClick}}" class="active">
                          <span class="value-primary">{{p[0]}}</
                              span>
                          <span class="badge badge-info">{{p[1]}}</
                              span>
                       </li>
                     </template>
                  </ul>
               </div>
         </div>
      </template>
    </div>
</template>
```

Listing 3.18: Faceted Search Template Code

Next (3.19), we have listed the implementation of the "Init()" function.

```
init: function () {

                this.params = this.params.split(',');

                for (var i= 0; i < this.params.length; i++)
                {
                    var selectorFunction = new Function ("d", "
                       return d."+ this.params[i] +".value;");
                    var dim = this.crossfilter.dimension(
                       selectorFunction);

                    var chart = dc.customFacetedSearch();
                    chart.dimension(dim);
                    chart.group(dim.group().reduceCount());
                    chart.polymer_element = this;
                    chart.dimensionName = this.params[i];

                    var values = [];
                    var val = dim.group().reduceCount().top(
                       Infinity);
                    for (var j= 0; j < val.length; j++)
                    {
                        values.push([val[j].key, val[j].value]);
                        chart.filter(val[j].key);
                    }

                    var element = [
                        this.params[i],
                        dim,
                        values,
                        chart
                    ];

                    this.dimensions.push(element);
                }//
            }
```

Listing 3.19: Faceted Search Script Code

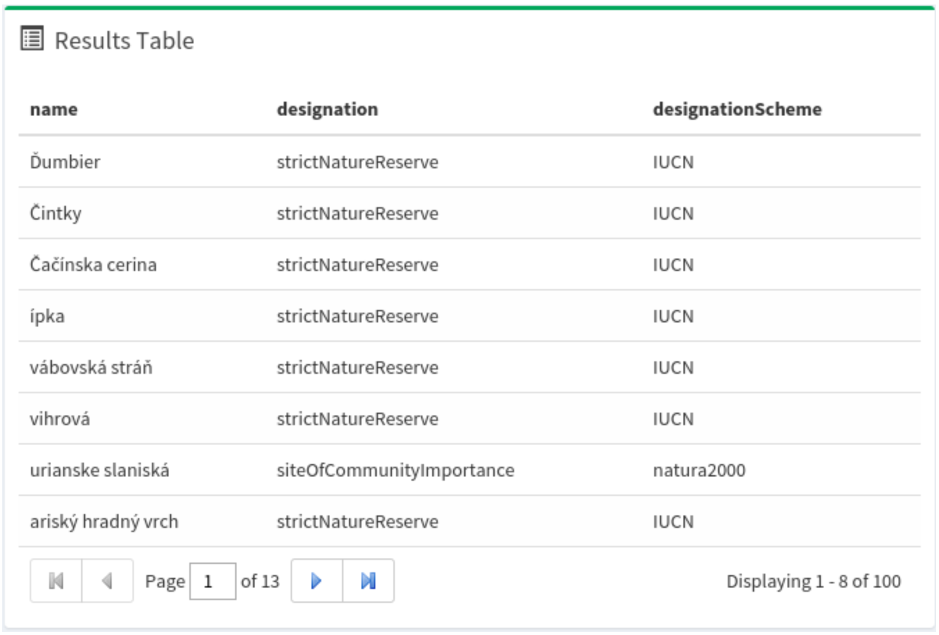
So, we are taking a list of parameters to take into account as the Polymer HTML tag parameters. Then, for each parameter, we create a Crossfilter's dimension, a group over that dimension, and a custom Dc.js chart (a script that inherits from Dc.js's customChart and overrides the onClick handler). Then, we explore all possible values with their counters and store a custom data structure into a ”dimensions” property. In the template tag, on the other hand, we iterate through this ”dimensions” and render them correctly.

This widget in particular fails at scaling to various thousands of data (even though Crossfilter can handle them easily), slowing the execution. This effect is not observed in chapter 5 (Evaluation), as we use a dataset that is not long enough to reproduce it. It would be possible in theory to optimize the code to get a sensible performance improvement when handling multiple thousands of data.

3.4.2.2 Results Table

This widget will be a table capable of rendering our current results and filter an individual element when we click on it. In Sefarad 1.0 we had a similar widget, done from datatables jQuery plugin. In this case, we cannot port that widget directly due to the malfunction of jQuery plugins inside Polymer elements.

We chose to adapt an existing Polymer object, implementing a Dc.js chart inside and the clickable functionality. The Polymer element is Steven skelston's ”sortable-table” element [15]


Results Table

name	designation	designationScheme
Ďumbier	strictNatureReserve	IUCN
Čintky	strictNatureReserve	IUCN
Čačínska cerina	strictNatureReserve	IUCN
ípka	strictNatureReserve	IUCN
vábovská stráň	strictNatureReserve	IUCN
vihrová	strictNatureReserve	IUCN
urianske slaniská	siteOfCommunityImportance	natura2000
ariský hradný vrch	strictNatureReserve	IUCN

Page 1 of 13

Displaying 1 - 8 of 100

Figure 3.8: Results Table Widget

The HTML code of the results table makes extensive use of the data binding and repeat templates features of Polymer. We can see an extract of this code that illustrate this mechanisms in 3.20.

```
<template if="{{!record.editMode}}" ref="{{rowTemplate}}" bind>
  <template repeat="{{column in columns}}" bind>
    <template ref="{{column.cellTemplate || cellTemplate}}" bind="
        {{record.fields[column.name]}}">
      <td class="column-{{column.name}}
        {{sortColumn == column.name && sortDescending ? 'sorted-
            column-desc' : ''}}
        {{sortColumn == column.name && !sortDescending ? 'sorted-
            column-asc' : ''}}">
        <template if="{{column.name == 'dbpediaLink'}}"><a target="
            _blank" href="{{value}}">{{value}}</a></template>
                <template if="{{!(column.name == 'dbpediaLink')}}"
                    >{{value}}</template></td>
    </template>
  </template>
</template>
```

Listing 3.20: Results Table HTML Code

The most noticeable change in the code has been the inclusion of filter effects in the table event handlers (3.21).

```
selectedChanged: function(a,val){
      if(val){
        if(this.isArray(val)){
          if(!this.multiSelect) this.multiSelect = true;
        }else{
          if(this.multiSelect) this.multiSelect = false;
        }
                this.dcChart.filter(val.name); //toggle filter
                dc.redrawAll();
      }
    }
```

Listing 3.21: Results Table Modifications Code

As we can see, we filter by the data's "name" property by default (although it can be abstracted) each time an element is clicked.

3.4.2.3 Pie Chart

This widget is the straight implementation of the Dc.js pie-chart. It follows all rules described in their documentation, and all we have to do is to expose its parameters to the outside via the custom HTML tag parameters.

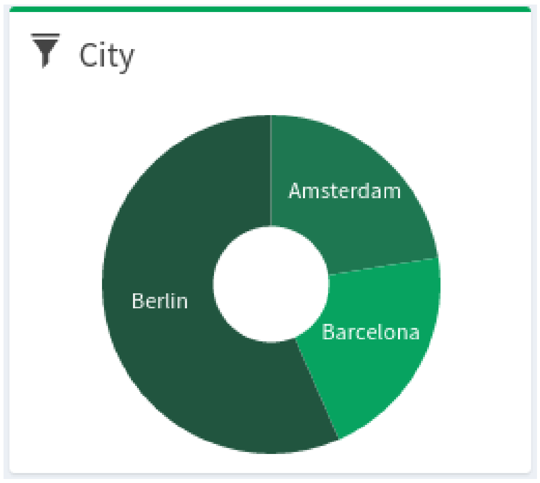


Figure 3.9: Pie Chart Widget

All we have to take care about is the problems between Polymer and Dc.js when styling and taking divs, as described in the appendixes.

The next block of code (3.22) illustrates the creation of the chart, where we list the HTML template and the init function.

```
<template>
  <!-- dc style -->
  <link rel="stylesheet" type="text/css" href="dc.css" media="
     screen" />
  <div id="chart"></div>
</template>

[...]

init: function () {
   var p = this.param;
   this.dimension = this.crossfilter.dimension(function(d){return d
      [p];});
   this.group = this.dimension.group().reduceCount();
   this.chart = dc.pieChart("#chart");
   this.chart
           .ordinalColors(["#1E7751","#07A360","#21553F","#25342E"
              ])
           .width(175).height(175)
           .dimension(this.dimension)
           .group(this.group)
           .innerRadius(30)
           .root()[0][0] = this.$.chart;
    this.$.chart.classList.add("dc-chart");
}
```

Listing 3.22: Pie Chart Code

As further improvements, there are still some parameters that could be abstracted from the initialization and set from the outside of the widget.

3.4.2.4 Bar Chart

This widget is another integration of an existing Dc.js chart into a Polymer element. This bar chart is documented in the Dc.js website.

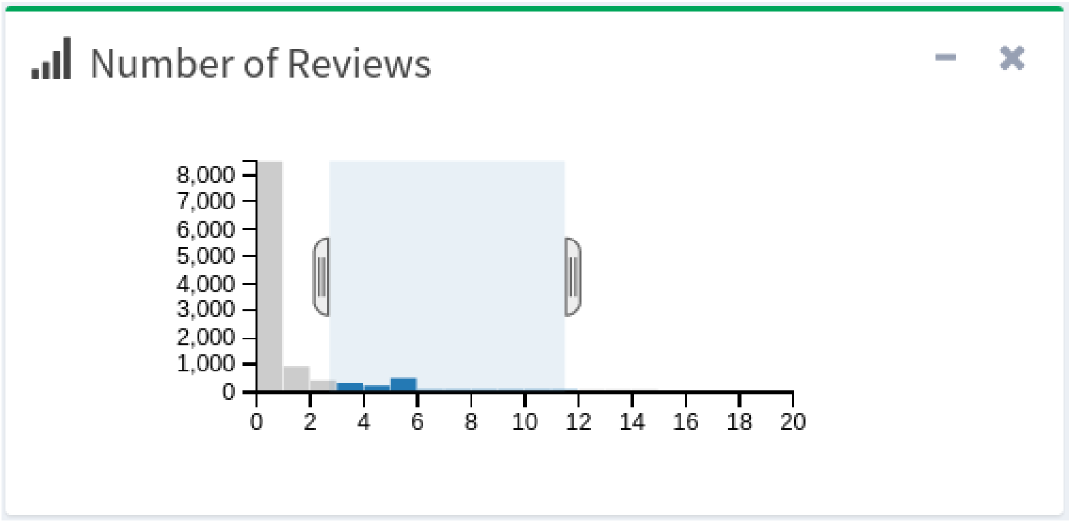


Figure 3.10: Bar Chart Widget

We apply the same integration rules that with the pie chart widget (3.23).

```
<template>
  <!-- dc style -->
  <link rel="stylesheet" type="text/css" href="dc.css" media="
      screen" />
  <div id="chart"></div>
</template>

[...]

init: function () {

                  var p = this.param;
                  this.dimension = this.crossfilter.dimension(
                      function(d) {
                       return d[p].value;
                  });
                  this.group = this.dimension.group().reduceCount();
                  this.chart  = dc.barChart("#chart");
                  var _width = parseInt(this.width);
```

```
                    this.chart
                            .width(_width).height(150)
                            .dimension(this.dimension)
                            .group(this.group)
                            .x(d3.scale.linear().domain([this.xMin,
                                this.xMax]))
                            .gap(0.0005)
                            .yAxisLabel("")
                            .xAxisLabel(this.xLabel)

                    .root()[0][0] = this.$.chart; //we pass explicitly
                        the root div to Dc.js
                    this.$.chart.classList.add("dc-chart"); //add the
                        dc-chart to allow Dc.js to style the chart;
                }
```

Listing 3.23: Bar Chart Code

3.4.2.5 Bubble Chart

As the previous one, this is a straight implementation of the Dc.js bubble chart into a Polymer element.

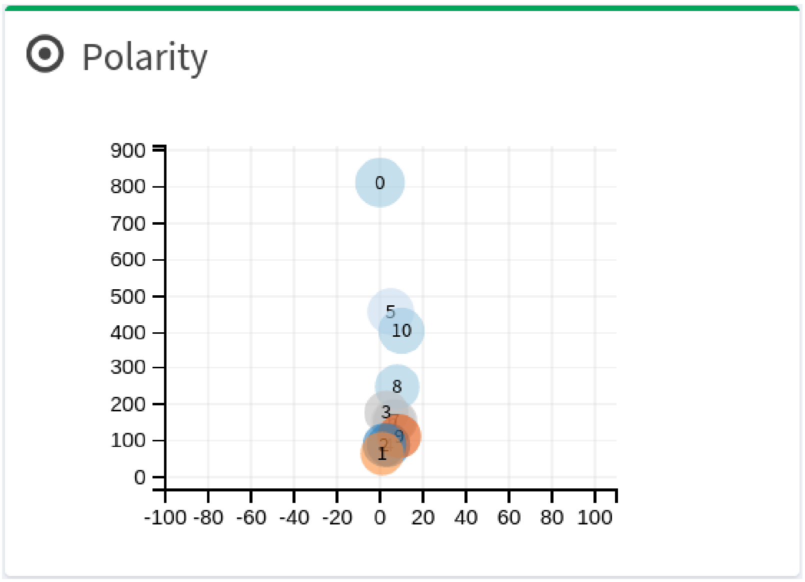


Figure 3.11: Bubble Chart Widget

The implementation is slightly larger that the previous two ones because there are more parameters to set, as color and radius functions (3.24).

```
<template>
  <!-- dc style -->
  <link rel="stylesheet" type="text/css" href="dc.css" media="
      screen" />
  <div id="chart"></div>
</template>

[...]

init: function() {

    var p = this.param;
    this.dimension = this.crossfilter.dimension(function(d) {
        return d[p].value;
    });
    this.group = this.dimension.group().reduceCount();
    this.chart = dc.bubbleChart("#chart");
    this.chart
        .transitionDuration(500)
        .margins({
            top: 10,
            right: 50,
            bottom: 30,
            left: 40
        })
        .width(300).height(200)
        .dimension(this.dimension)
        .group(this.group)
        .colorAccessor(function(d) {
            return d.value;
        })
        .keyAccessor(function(d) {
            return d.key;
        })
        .valueAccessor(function(d) {
            return d.value;
        })
        .radiusValueAccessor(function(d) {
            return d.value;
        })
        .maxBubbleRelativeSize(0.5)
        .renderHorizontalGridLines(true) // (optional) render
           horizontal grid lines, :default=false
        .renderVerticalGridLines(true)
        .x(d3.scale.linear().domain([0, this.xMax]))
        .y(d3.scale.linear().domain([0, this.yMax]))
        .r(d3.scale.linear().domain([0, this.radius]))
        .elasticY(true)
        .elasticX(true)
        .yAxisPadding(100)
        .xAxisPadding(100)
        .root()[0][0] = this.$.chart; //we pass explicitly the root
            div to Dc.js
    this.$.chart.classList.add("dc-chart"); //add the dc-chart to
        allow Dc.js to style the chart

}
```

Listing 3.24: Bubble Chart Code

The possibilities of this chart could be explored further in future dashboards, including effects such as combining multiple parameters in its distinct variables (radius, color, x and y axis).

3.4.2.6 Number Chart

A number chart is a widget capable of display the total number of filtered elements in a concrete dimension or the number of distinct values of an specific parameter remaining after filtering.

You can choose between the two functionalities through the type parameter on the tag interface. The two possibilities are ”param” and ”value”. When ”value” mode is selected, you must specify which value you want to count, and the result will render the total count of features that match that particular value. When ”param” modo is selected, the widget will count how many distinct values can be found among the filtered data.

Moreover, we have designed two different skins for this widget:

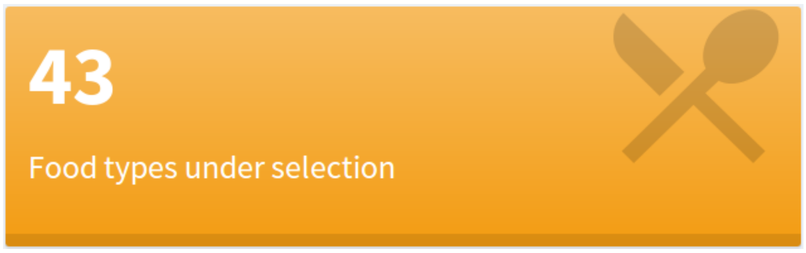


Figure 3.12: Number Chart Widget, skin 1

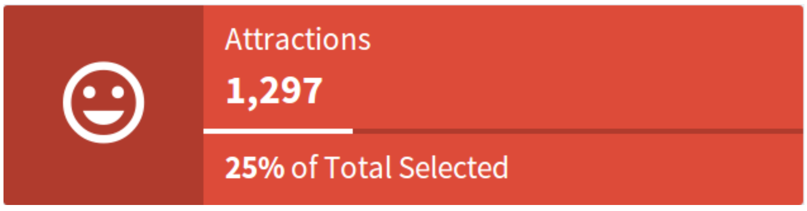


Figure 3.13: Number Chart Widget, skin 2

The first skin appears in the "restaurants" demo, and is a straight implementation of the AdminLTE predefined number chart. In this case, the only element driven by the Polymer element is the text with the actual number, styled with the same sheets than the rest of the AdminLTE widgets.

This fact means that, in this case, it doesn't exist a complete encapsulation of the widget inside Polymer. We can't inject all content in the Polymer element due to some animated interaction provided by the AdminLTE code which, in case of being everything inside Polymer, would not be able to reach the content inside the shadow DOM.

The second skin suffers from the same issue but, in its case, it has control over two elements outside of its Polymer, the percent span and the progress bar component.

In code (3.25) we will try to set those two external elements (in the case of the second skin) and catch the error if none of them are found (in the first case).

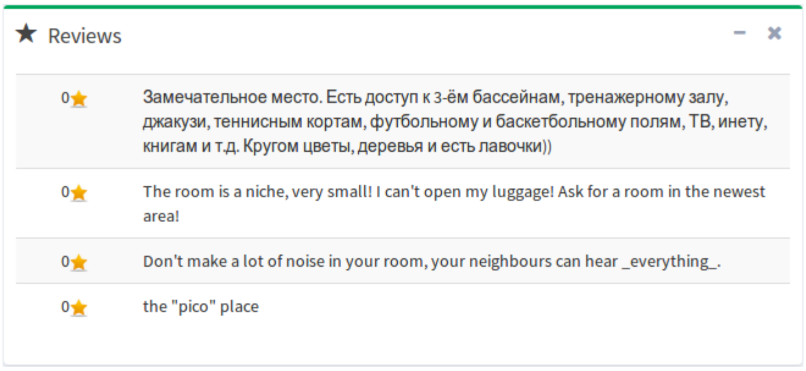
```
<template>
  <!-- dc style -->
  <link rel="stylesheet" type="text/css" href="dc.css" media="
      screen" />
  <div id="chart"></div>
</template>

 [...]

 init: function() {

   var vCount = this.valueToCount; //HTML custom tag parameter
     var val = dim.group().all();
     for (var k= 0; k < val.length; k++)
     {
         if(val[k].key == vCount) this.MAX = val[k].value;
     }
     this.chart = dc.numberDisplay("#chart");

     if(this.type == "param") //first widget type
     {
         var v = this.param;
         var _MAX2 = this.dimension.group().all().length;
         this.chart
                 .valueAccessor(function(d) {
                    var aux = [];
                    var val = dim.group().all();
                    for (var k= 0; k < val.length; k++) {

                        if(val[k].value != 0) aux.push(val[k]);

                    }
                    var percent = (aux.length/_MAX2)*100;
                    percent = Math.round(percent);
                    try {
                        document.querySelector(".percent-" + v).
                            textContent = percent + "%";
                        document.querySelector(".progress-bar-" + v
                            ).style.width = percent + "%";
                    } catch (e){
                        //no progress bar and percent text found.
                    }
                    return aux.length;
                })
                .group(this.group)
                .root()[0][0] = this.$.chart; //we pass explicitly
                   the root div to Dc.js
    }
    if(this.type == "value") //second widget type
    {
        var v = this.valueToCount;
        var _MAX = this.MAX;
        var aux = 0;
        this.chart
                .formatNumber(d3.format(",3d"))
                .valueAccessor(function(d) {

                    var val = dim.group().all();
                    for (var k= 0; k < val.length; k++) {
                        if(val[k].key == v){
                            aux = val[k].value;
                        }
                    }
                    var percent = (aux/_MAX)*100;
                    percent = Math.round(percent);
                    try {
                        document.querySelector(".percent-" + v).
                            textContent = percent + "%";
                        document.querySelector(".progress-bar-" + v
                            ).style.width = percent + "%";
                    } catch (e){

                    }
                    return aux;
                })
                .group(this.group)
                .root()[0][0] = this.$.chart; //we pass explicitly
                    the root div to Dc.js
    }
    this.$.chart.classList.add("dc-chart"); //add the dc-chart to
        allow Dc.js to style the chart

}
```

Listing 3.25: Number Chart Code

To conclude we could give this widget another iteration to try encapsulating all its code so the final result is clean and reusable, as it is aimed in Polymer's philosophy.

3.4.2.7 Reviews (Custom Widget example)

This widget respond to a specific application need in the "tourpedia" places demo. In this demo, we needed to query a web service in case that only one element remained unfiltered (e.g. the user has selected one place in). Therefore, this widget needs to access to the Dc.js filters in order to check how many elements remain. The result of the web service query is the rendered in a table with the help of Polymers iterator HTML constructor that binds template shapes to data. Once the data has been retrieved and rendered the widget seems like this:



0	Замечательное место. Есть доступ к 3-ём бассейнам, тренажерному залу, джакузи, теннисным кортам, футбольному и баскетбольному полям, ТВ, инету, книгам и т.д. Кругом цветы, деревья и есть лавочки))
0	The room is a niche, very small! I can't open my luggage! Ask for a room in the newest area!
0	Don't make a lot of noise in your room, your neighbours can hear _everything_.
0	the "pico" place

Figure 3.14: Reviews Custom Widget

In order to include this new functionality, we have overwritten the valueAccessor() function of the regular numberDisplay Dc.js widget. In a wide view, it counts how many features are filtered each time it needs to access the value to render. If it is equal to one, an asynchronous petition is fired. If it is not, it destroys all the HTML content and renders an informative message.

In the next block of code (3.26) we detail how the implementation is done. The HTML part, the template, implements only a div that we later fill from this script code.

```
.valueAccessor(function (d) {
    var val = dim.group().all();
    var aux = [];
    for (var k = 0; k < val.length; k++) {
        if (val[k].value != 0) aux.push(val[k]);
    }
    if (aux.length == 1) {
        var _id = aux[0].key;
        $.getJSON("http://tour-pedia.org/api/getReviewsByPlaceId?
            placeId=" + _id, function (data) {
            parsedData = data;
```

```
                if (parsedData.length == 0) {
                    tableDiv.innerHTML = "No reviews of this place";
                } else {
                    tableDiv.innerHTML = "";
                    var tbl=document.createElement('table');
                    tbl.className += "table table-striped";
                    tbl.style.width='100%';
                    for(var i = 0; i < parsedData.length; i++){
                        var tr = tbl.insertRow();
                        var td = tr.insertCell();
                        td.appendChild(document.createTextNode(
                            parsedData[i].text));
                        td.style.width = "85%";
                        td = tr.insertCell();
                        td.appendChild(document.createTextNode(
                            parsedData[i].rating));
                        var img = document.createElement("img");
                        img.src = "https://cdn4.iconfinder.com/data/
                            icons/pretty_office_3/256/Star-Full.png";
                        img.style.height = "15px";
                        td.style.width = "15%";
                        td.style.textAlign = "center";
                        td.appendChild(img);
                    }
                    tableDiv.appendChild(tbl);
                }
            });
    } else {
        tableDiv.innerHTML = "Select one place";
    }
    return 0;
})
```

Listing 3.26: Reviews Widget Code

This widget comes to illustrate how you can use Polymer components to fulfil a specific dashboard need with little effort.

3.4.2.8 Map

This map is the integration of a leaflet map with a Dc.js base chart, all working as a Polymer component.

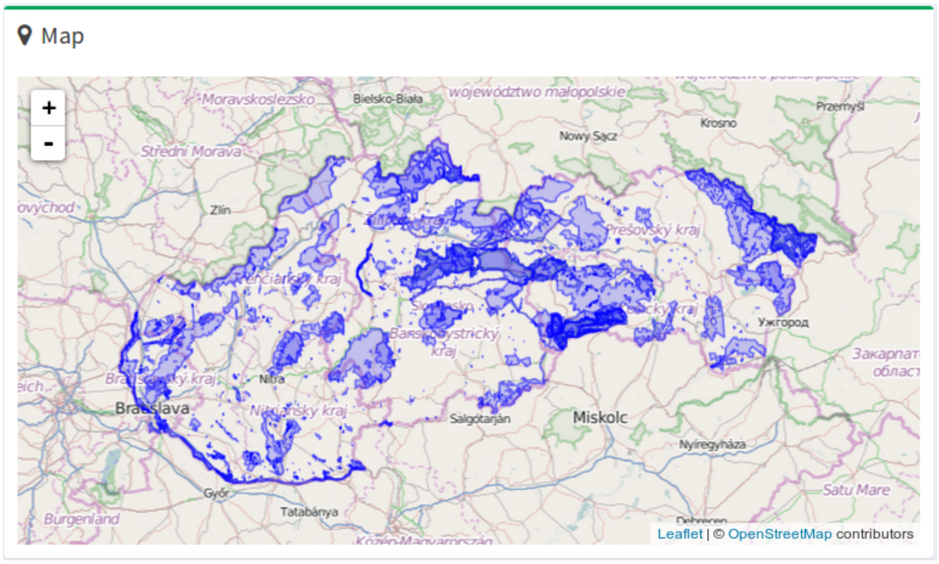


Figure 3.15: Polygons Map Widget

First, we use the geoJSON layer that leaflet features to render the polygonal data in the correct coordinates. Then we have to transform this map into a Dc.js chart.

The key to solve this complex task is to rewrite the Dc.js's base chart code so it can handle all filter changes and update the map accordingly. We can take advantage of leaflet's click feature that enables us to identify which polygon we have selected, and render a pop-up over it.

Below (3.27) we find the code of our custom Dc.js chart.

```
var selectFilter = function(e) {
    if (!e.target) {
        return;
    }

    var filter = e.target.key;
    dc.events.trigger(function() {

        _chart.filter(filter);
        if (_chart.filters().indexOf(filter) != -1) {
            _chart.mustReDrawBool(false);
        } else {
            _chart.map().closePopup(); //avoid showing the popup,
                cause we are deselecting.
        }

        dc.redrawAll(_chart.chartGroup());
    });
};
```

Listing 3.27: Map Widget onClick Code

We are using a variation of Boyan Yurukov's implementation, with a great performance boost: In Yurukov's widget each time we update the data, the map is destroyed and reconstructed while in our case (3.28) we play with the styling so all we have to do is highlight the selected ones and hide the filtered without more workload.

```
var _featureStyle = function(feature) {
    var options = _chart.featureOptions();
    if (options instanceof Function) {
        options = options(feature);
    }
    options = JSON.parse(JSON.stringify(options));
    var v = _dataMap[_chart.featureKeyAccessor()(feature)];
    if (v && v.d) {
        options.color = '#B9A081';
        options.fillColor = '#B9A081',
        options.weight = 1.5;
        options.opacity = 0.6;
        options.fillOpacity = 0.4;
        if (v.d.value == 1) {
            options.color = 'blue';
            options.fillColor = 'blue',
            options.weight = 1.5;
            options.opacity = 0.5;
            options.fillOpacity = 0.2;
        }
        if (_chart.filters().indexOf(v.d.key) !== -1) { //selected
           in this chart
            options.color = '#FD8F00';\
            subsection {
                Polygons Map
            }
            options.fillColor = '#FD8F00',
            options.weight = 2;
            options.opacity = 0.6;
            options.fillOpacity = 0.4;
        }
    }
    return options;
};
```

Listing 3.28: Map Widget Styling Code

Then we can assign pop-ups to each feature pre-processing at data load (3.29).

```
var processFeatures = function (feature, layer) {
    var v = _dataMap[_chart.featureKeyAccessor()(feature)]
    if (v && v.d) {
        layer.key = v.d.key;
        if (_chart.renderPopup()) {
            layer.bindPopup(_chart.popup()(v.d, feature));
        }
        if (_chart.brushOn()) {
            layer.on("click", selectFilter);
        }
    }
};
```

Listing 3.29: Map Widget Pop-up Code

In a new study case, "tourpedia", we need to render thousands of locations at the same time. This scale force us to look for an optimized solution as clustering techniques. In this context we have developed an adaptation of this widget that, over the previous base, implements the "markercluster" leaflet plugin. This widget maintains all the functionality of the polygonal map but the visual appereance.

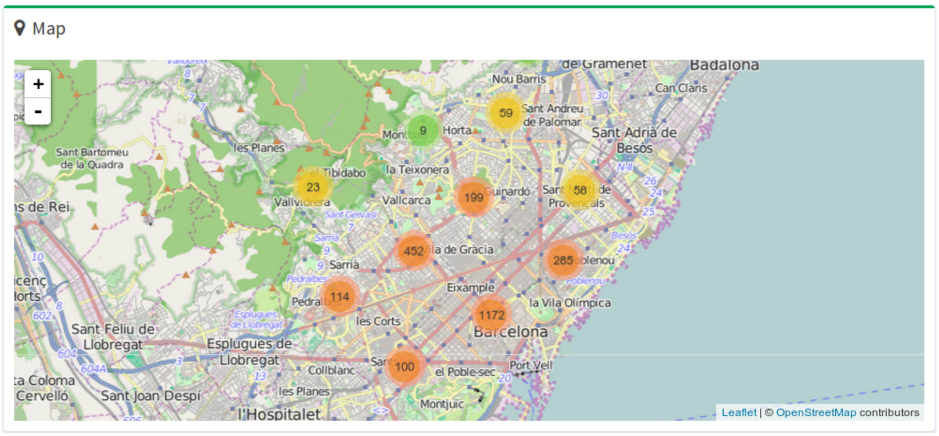


Figure 3.16: Marker Map Widget

3.5 Summary

In this chapter we have reviewed in depth Sefarad's structure and functionality. We have seen in first place its architecture, explaining the details of each module. After that we have defined the interaction model, or how is it supposed to be used by the final user. Then, we have listed and explained the widgets structure and the implementation details of each one of those which we have used in the three case studies. Finally we have go through the process of designing and implementing a new dashboard from an empty template.

CHAPTER 4

Case Study

This chapter covers the process of implementation of Sefarad 2.0 to three different datasets. We will explain the origin of the data and their structure, analysing its virtues and lacks. Then we will provide a detailed description of the analysis widgets for each one.

4.1 Introduction

In this chapter we present the three use cases we have applied Sefarad 2.0 to.

The first one is the reconstruction of the Sefarad 1.0 demonstration for the Smart Open Data project. This case will be centred in the development of tools for rendering and filter polygonal areas. We will analyse the facets lackness of this dataset, giving ideas of how to solve it.

The second case also shares a dataset with Sefarad 1.0 and aim to develop a more complete analysis dashboard taking advantage of the data's great quantity of information. We will go through the designing process and give our conclusions.

The last case is the most complete and unify the quality of data with the quantity, aiming to demonstrate the power of crossfilter and dc at the same time that it offers a well designed graphical interface.

4.2 Slovak Demo: Smart Open data

4.2.1 Origin

This case study is implemented in the context of Smart Open Data. This European project aims to create a Linked Data infrastructure fed by public and freely available data resources for biodiversity and environment protection and research in rural and protected areas or national parks.

This first case study served as a design base at the first development stages. It is mostly a redo of the last project and the final product must implement the same functionality in a smarter and cleaner way.

We will work with the dataset provided by the Slovak Environmental Agency (SEA[1]) about harmonised protected sites dataset according to INSPIRE[2] Data Specification on Protected Sites – Guidelines through WFS service interface.

SEA provided source files for Geoserver workspaces related to Slovak protected sites feature type. Source files can be downloaded from SEA website[3].

The data has the following fields and information values, which will be used to test faceted browsing and geo filtering with ECQL[4].

[1]http://www.sazp.sk/
[2]http://inspire.ec.europa.eu/
[3]http://inspire.geop.sazp.sk/geoserver/www/eenvplus/ps_harmonisation.tgz
[4]http://docs.geoserver.org/latest/en/user/filter/ecql_reference.html

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
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 4.1: Data INSPIRE designation

In this case, we store the information into a local Fuseki database and we query it from Sefarad. The Query we execute 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 4.1: Slovak Demo SPARQL

4.2.2 Structure and pre-process

This dataset contains a total of 1645 features. Each feature retrieved as a JSON element looks like the following:

```
{
  "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 4.2: Slovakia results JSON example

Lets break this code into pieces, taking in consideration only relevant information:

- **Linked Data URI (res)**: Each element has a unique URI direction following Linked Data directives.
- **feature Well Known Text (fWKT)**: Contains a multi-polygon element describing the geometry of each feature.
- **name**: The area's name.
- **Designation**: Abstract base type for code lists containing the classification and designation types under different schemes. Some of these designation and classification lists are closed (for example, Natura2000), while some change regularly. Defined under the INSPIRE directive.

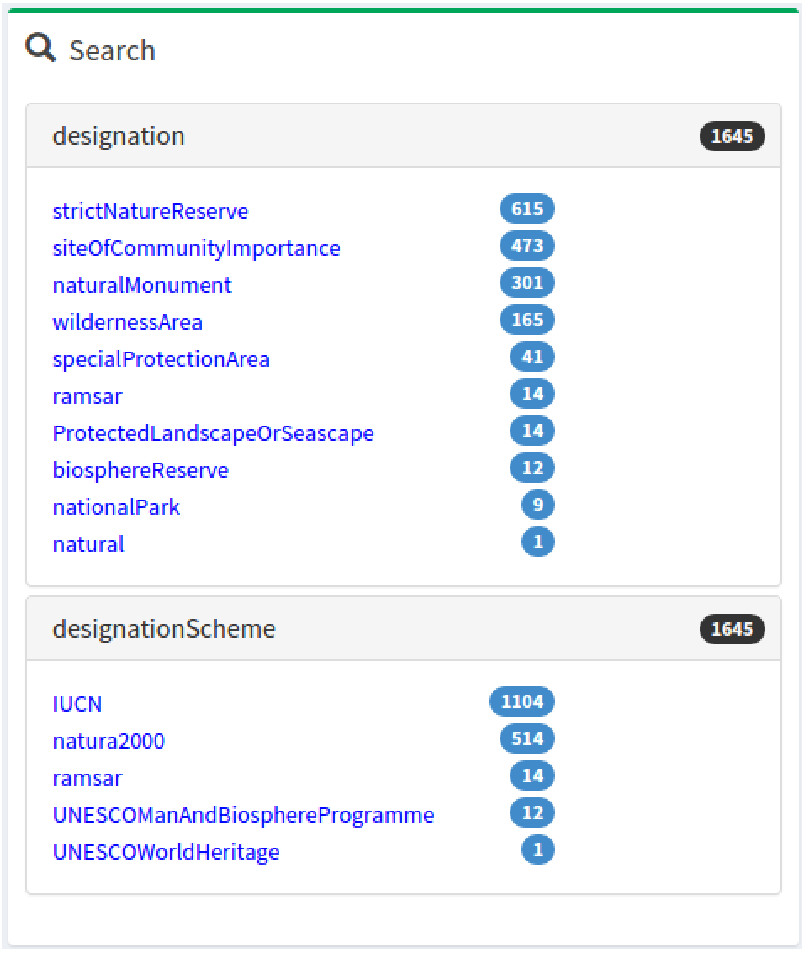
  The possibilities are: strict nature reserve, site of community importance, natural monument, wilderness area, special protection area, ramsar, protected landscape or seascape, biosphere reserve, national park, natural.
- **Designation Scheme**: The scheme used to assign a designation to the Protected Sites. Schemes may be internationally recognised (for example, Natura2000 or the Emerald Network schemes), or may be national schemes (for example, the designations used for nature conservation in a particular Member State). Defined under the INSPIRE directive.

  The possibilities are: IUCN, natura 2000, ramsar, UNESCO man and biosphere program, UNESCO world heritage.

4.2.3 Analysis Design

As it can be observed, this dataset offers a limited number of parameters to filter and render, so we have focused on the three most representative of them: designation, designation scheme, and polygonal data.

At first place, on the top left of the dashboard, we have placed a faceted search widget for the designation and designation scheme:


Search

designation	1645
strictNatureReserve	615
siteOfCommunityImportance	473
naturalMonument	301
wildernessArea	165
specialProtectionArea	41
ramsar	14
ProtectedLandscapeOrSeascape	14
biosphereReserve	12
nationalPark	9
natural	1

designationScheme	1645
IUCN	1104
natura2000	514
ramsar	14
UNESCOManAndBiosphereProgramme	12
UNESCOWorldHeritage	1

Figure 4.1: Slovak Faceted Search

At its side, we have placed the polygonal map, rendering all polygons at once and implementing pop-ups with the features name, selection on click and interaction with other widgets:

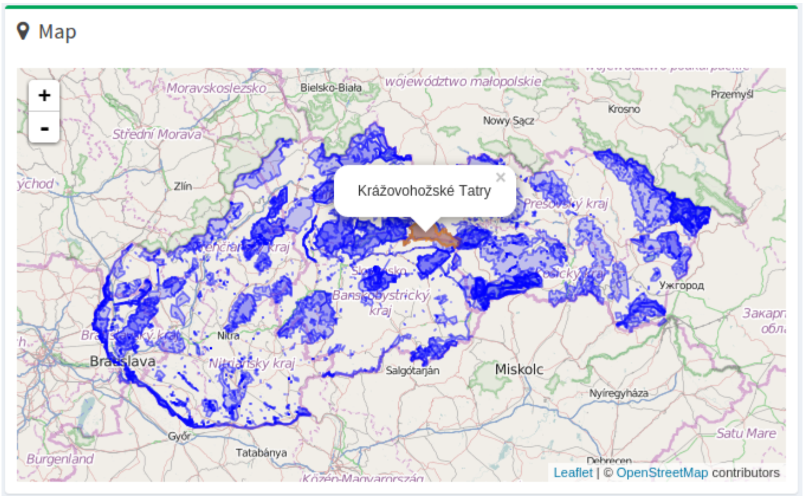
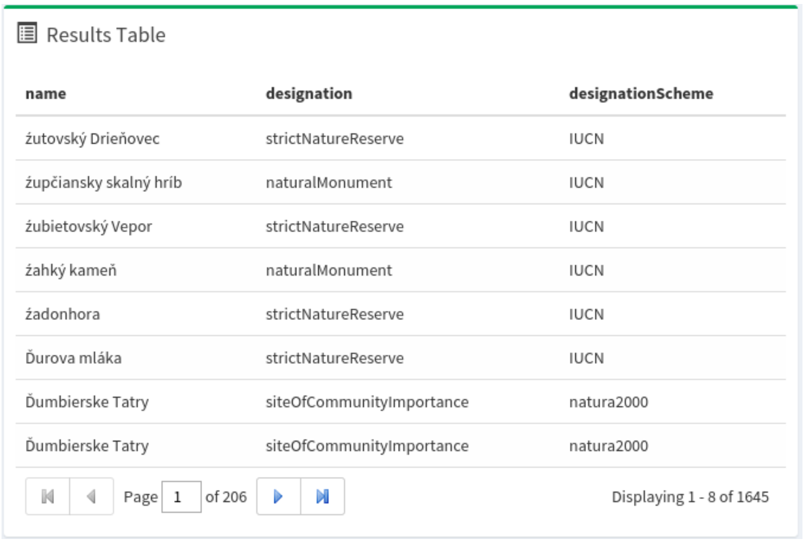


Figure 4.2: Slovak Map

Below the map we have a results table, capable of filtering on double click and displaying the name, designation and designation scheme of the filtering results.



Results Table

name	designation	designationScheme
žutovský Drieňovec	strictNatureReserve	IUCN
župčiansky skalný hríb	naturalMonument	IUCN
žubietovský Vepor	strictNatureReserve	IUCN
žahký kameň	naturalMonument	IUCN
žadonhora	strictNatureReserve	IUCN
Ďurova mláka	strictNatureReserve	IUCN
Ďumbierske Tatry	siteOfCommunityImportance	natura2000
Ďumbierske Tatry	siteOfCommunityImportance	natura2000

Page 1 of 206 Displaying 1 - 8 of 1645

Figure 4.3: Slovak Results Table

At last, we have set two pie chart widgets to offer the same filtering functionality of the faceted search over designation and designation scheme but in a graphical way:

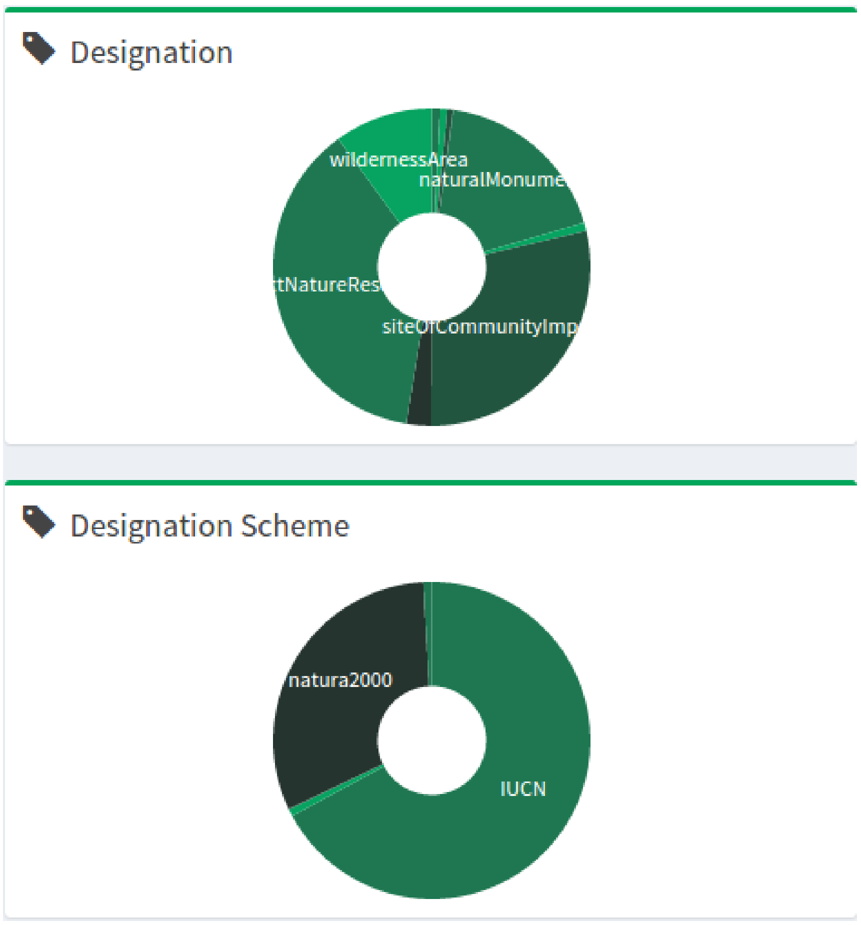


Figure 4.4: Slovak Pie Charts

4.2.4 Conclusions

As we can see, this dataset provides a lot of features with great polygonal detail and it success at its map demonstration purpose.

However, it doesn't contain much useful information about the features. One possible solution to apply in future iterations would be look for facets for the data in external sources, via federated SPARQL queries or web requests at post-processing time.

4.3 Restaurants Demo

4.3.1 Origin

The restaurants demo is an update of the one developed for Sefarad 1.0. In the past, we found this dataset online and used it to demonstrate the capabilities of the recently developed marker map. At the end, the demo was declined for the final project, as we wanted to show polygons, and this restaurants data doesn't have them. This dataset is a compilation of restaurants from Madrid.

After the implementation of the Smart Open Data project demo, we started looking for a dataset that had enough number of facets and values in order to show the true power of dc.js in our widgets and inspire us for new widget designs.

The data has been obtained from different government website madrid.org[5] and the restaurants online browser yelp[6].

The dataset can be queried at: ***http://demos.gsi.dit.upm.es/fuseki/restaurants/query***

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*.

[5]http://www.madrid.org/nomecalles/

[6]http://www.yelp.com/madrid

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 4.3: Restaurants SPARQL Query

4.3.2 Structure and pre-process

The data is a collection of 104 features with a rich set of facets. This is the JSON asociated to one feature on the query response:

```
"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 4.4: Restaurants results JSON example

So in the data we can found:

- **Linked Data URI**: Due to the fact of being organized in a RDF format following the Linked Data Directive, each element has a unique URI direction.
- **DbPedia Link**: Each restaurant is linked by name to its corresponding dbPedia page (if it exists).
- **District**: The restaurants are geolocated among all Madrid's area, belonging to only one district. The possible districts are: Chamberí, Usera, Latina, Retiro, Arganzuela, Tetuán, Carabanchel, Salamanca and Vicálvaro.
- **Food type**: Each restaurant is tagged with its most noticeable type of food. There are 43 different types of food among all the restaurants from Italian, Middle eastern and Beer bars... to Juice Bars and Smoothies.
- **Latitude and longitude**: we can use them to geo-locate the restaurants and render them in a map. All restaurants are within the bounds of Madrid city.
- **Name**: The most important facet, since it is the one that the user will look for applying filters afterwards.
- **Stars**: Each restaurant is rated by its quality with a rating from 0 to 5 starts. Besides that, our dataset doesn't include any restaurant with a rating below 3, being the ocurrences: 3, 3.5, 4, 4.5 and 5.
- **TakeOut**: This boolean facet shows if the restaurant offers the possibility of taking away the food to eat it elsewhere.
- **Price**: Each feature has a price given a a range. We offer this raw information at the data table, but for the sake of simplicity when filtering we have applied some pre-processing, so we assign to each feature a price that is computed as the mean of the extreme values of that range.
- **Reservations**: This is another boolean property that points if the restaurant can handle reservations for their clients.

4.3.3 Analysis Design

We have developed some widgets around this dataset trying to offer an example of a clear and useful interface for filtering the data and retrieve interesting results.

At the top of the page we can find four number charts (implemented as the first skin) to quickly put some interesting information at sight:

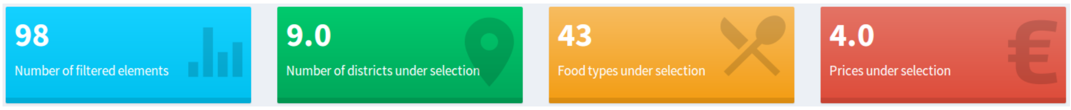

98
Number of filtered elements
9.0
Number of districts under selection
43
Food types under selection
4.0
Prices under selection

Figure 4.5: Number Charts

The first chart counts how many restaurants are left after the current filters. It is a number chart set to count a parameter different values (in this case, the parameter is the name, so it is counting how many different names are there).

The second counts how many districts can be found among the current data selection. As the previous one, this chart is set to count the different values of a parameter (in this case, district).

"Food types under selection" and "Prices under selection" are set the same way, so they are all counting distinct values of a given parameter.

With this four charts we can given a wide impression of how accurate and close is our selection.

Next we have two pie charts handling filters for the "reservations" and "takeout" parameters:

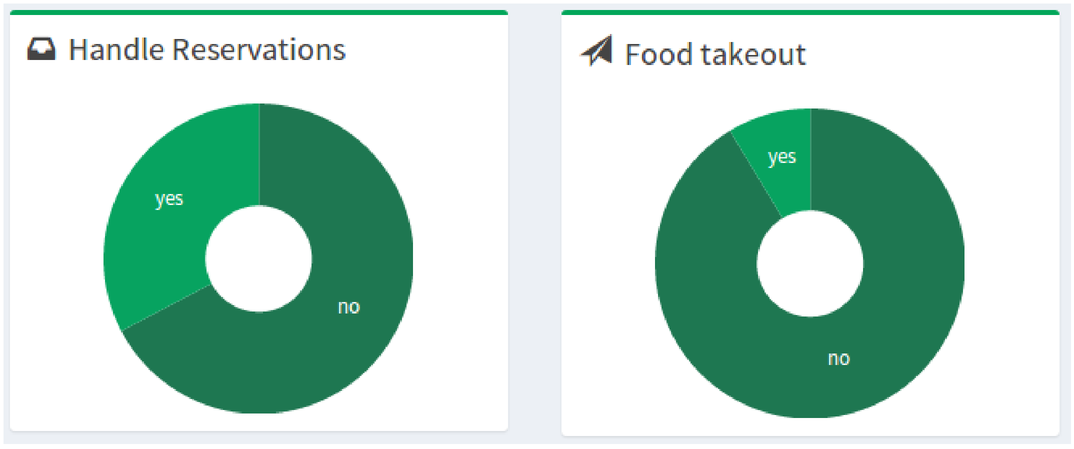


Figure 4.6: Pie Charts

We chose the pie chart for these two parameters due to their boolean nature, so it is very easy to understand what they do with a simple eyesight.

At their side, we have two different widgets for the same property, the price:

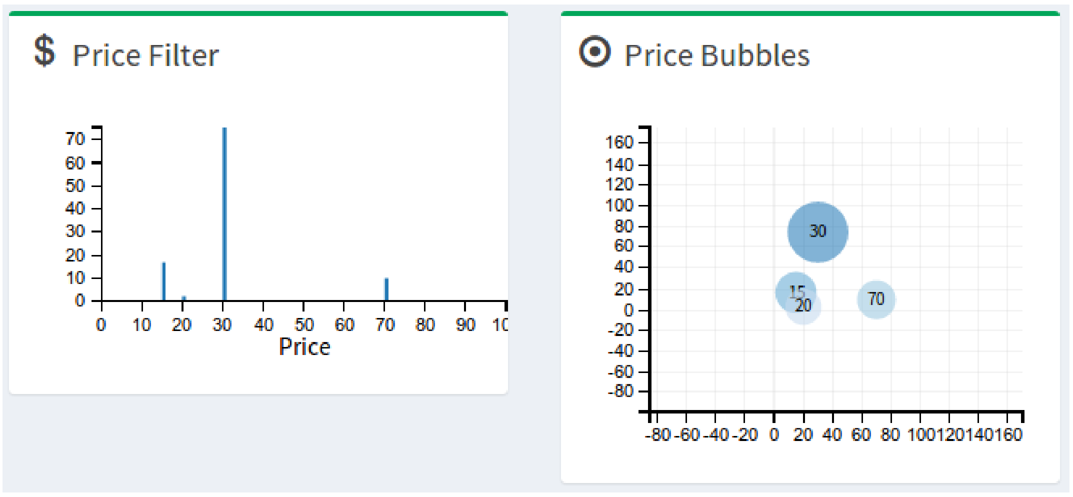
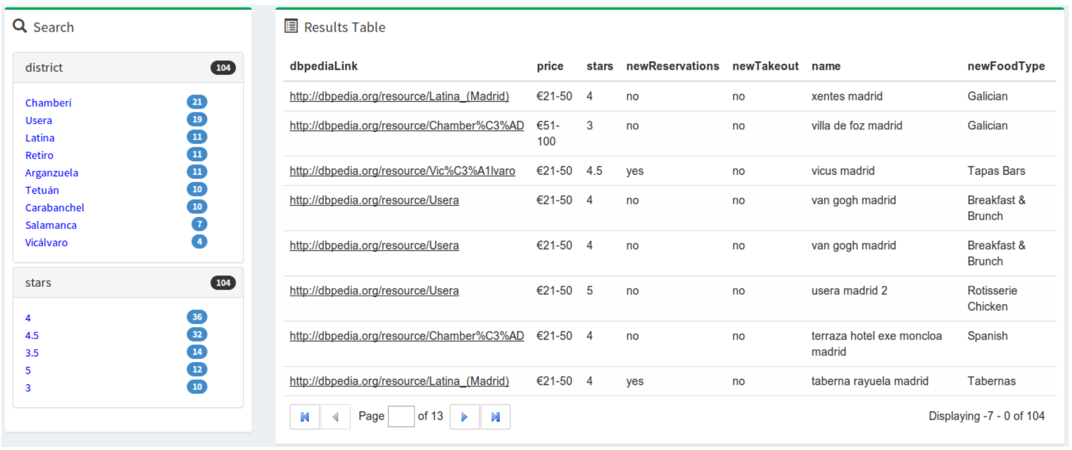


Figure 4.7: Price Bar and Bubble Charts

The first one presents the price as a bar chart and able us to filter it in a given range graphically moving its extremes.

The second one gives a more visual picture of which price is more present among the data taking advantage of size, color and position of the bubbles. All these widget properties are set to scale with the same price property.

The last row are the two main widgets of this demo, the faceted search and results table widgets:



Search

district	104
Chamberí	21
Usera	19
Latina	11
Retiro	11
Arganzuela	11
Tetuán	10
Carabanchel	10
Salamanca	7
Vicálvaro	4

stars	104
4	36
4.5	32
3.5	14
5	12
3	10

Results Table

dbpediaLink	price	stars	newReservations	newTakeout	name	newFoodType
http://dbpedia.org/resource/Latina_(Madrid)	€21-50	4	no	no	xentes madrid	Galician
http://dbpedia.org/resource/Chamber%C3%AD	€51-100	3	no	no	villa de foz madrid	Galician
http://dbpedia.org/resource/Vic%C3%A1lvaro	€21-50	4.5	yes	no	vicus madrid	Tapas Bars
http://dbpedia.org/resource/Usera	€21-50	4	no	no	van gogh madrid	Breakfast & Brunch
http://dbpedia.org/resource/Usera	€21-50	4	no	no	van gogh madrid	Breakfast & Brunch
http://dbpedia.org/resource/Usera	€21-50	5	no	no	usera madrid 2	Rotisserie Chicken
http://dbpedia.org/resource/Chamber%C3%AD	€21-50	4	no	no	terraza hotel exe moncloa madrid	Spanish
http://dbpedia.org/resource/Latina_(Madrid)	€21-50	4	yes	no	taberna rayuela madrid	Tabernas

Page [ ] of 13

Displaying -7 - 0 of 104

Figure 4.8: Faceted Search and Results Table

The faceted search filters by two parameters: district and stars. It would be interesting to add to this widget the parameter "foodType", but the idea was rejected due to the amount of food types (values of that parameter). However, it could be possible to include that parameter in the form of an accordion minimized (for the sake of a clean interface).

The table widget includes here the main parameters for all the selected features. It is worth mentioning that we are including here the links to dbpedia for each feature. They are functional, opening a new tab with the matching dbpedia page on click.

4.3.4 Conclusions

As we can notice this is a much richer dataset comparing to the Slovak data. We have more facets with a lot of values, enable us to be more creative in the interface design.

Besides, it fails at an important point: the quantity of features. 104 is a poor number for the capabilities of crossfilter-dc.js, in order to really test them we will have to expand this data or find a new one with more features.

4.4 Tourpedia Demo

4.4.1 Origin

After the two previous demonstrations we started looking for a dataset that has the goodnesses of both of them. At one hand we need quantity of features to test the capabilities of crossfilter and dc.js at the time of filtering huge amounts of data. On the other hand we look for data quality, a rich dataset which allows us to develop a data analysis dashboard that outputs a meaningful result.

In that quest we found Tourpedia. TourPedia is the Wikipedia of Tourism. It contains information about accommodations, restaurants, points of interest and attractions of different places in Europe. At the moment eight cities are covered: Amsterdam, Barcelona, Berlin, Dubai, London, Paris, Rome and Tuscany. However, they plan to extend the service to all the world. Data are extracted from four social media: Facebook, Foursquare, Google Places and Booking.

TourPedia provides two main datasets: Places and Reviews. Each place contains useful information such as the name, the address and its URI to Facebook, Foursquare, Google-Places and Booking. Reviews contain also some useful details ready for us to exploit.

TourPedia provides two methods to access data: through a Web API and a SPARQL engine. It is exposed through the SPARQL engine as a linked data node, which provides access to places. Reviews can only be accessed through web interface.

At first we downloaded the places dataset in JSON format (stored in Sefarad's own server), and lately we prepared a query to use their SPARQL endpoint. Due to its size, we have clamp to a handleable number of features, taking places from just 3 cities and 2 types of places: restaurants and attractions .

4.4.2 Structure and pre-process

The data is a collection of 12.000 features with a rich set of facets. This is the JSON asociated to one feature on the query response:

```
{
 "id":223771,
 "name":"American Hotel",
 "address":"Leidsekade 97, Amsterdam, Netherlands",
 "category":"accommodation",
 "location":"Amsterdam",
 "lat":52.363826,
 "lng":4.881369,
 "numReviews":5,
 "reviews":"http:\/\/tour-pedia.org\/api\/getReviewsByPlaceId?
    placeId=223771",
 "polarity":8,
 "details":"http:\/\/tour-pedia.org\/api\/getPlaceDetails?id
    =223771"
}
```

Listing 4.5: PLaces JSON example

This is the format taken from the downloaded JSON. The current response, queried from the SPARQL endpoint, has the same format that the rest of the demos. In this case, we have to transform the data format. That will be done in the main Sefarad script, after the data is retrieved and before spreading it to the widgets.

In these data, we can found the following information:

- **Id**: The unique id that each place has in their database.
- **Name**: Name of the place.
- **Address**: Detailed address of the place.
- **Location**: City in which the place is settled.
- **Latitude - longitude**: Coordinates used for rendering in the map.
- **Number of reviews**: Number of reviews stored in their online database.
- **Reviews direction**: URI of the web service where we can find the reviews asociated to this place.
- **Polarity**: Number from 0 to 10 meaning the positive or negative impression that can be inferred from the reviews of this site.
- **Details direction**: URI of the web service where we can find more facets of this place.

4.4.3 Analysis Design

We have analyse the information of this rich dataset the following way:

In the first row of widgets we have a marker map, containing all active results, and four number charts in their second skin. These are: total elements selected (parameter type), polarities under selection (parameter type), attractions (value type) and Accommodation (value type).

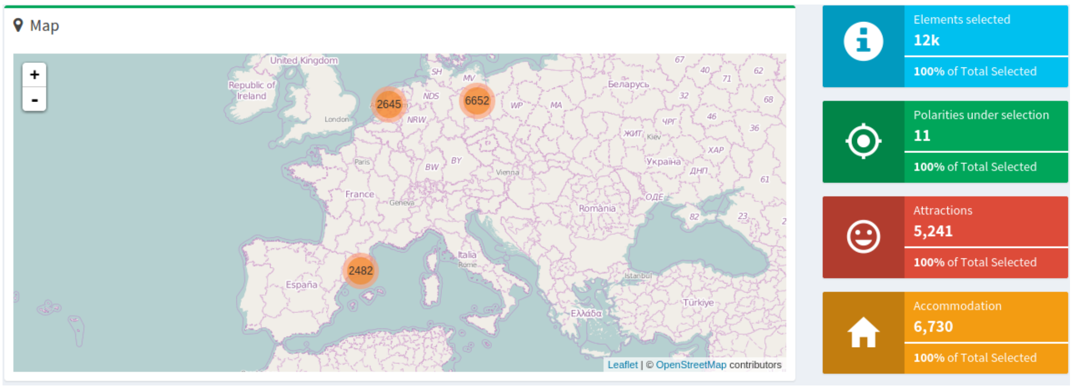


Figure 4.9: Map and Number Widgets

Next, we have a row of proper filters: two pie charts, for category and location parameters, one bar chart for the number of reviews, and a bubble chart for illustrating the weight of each polarity in the dataset:

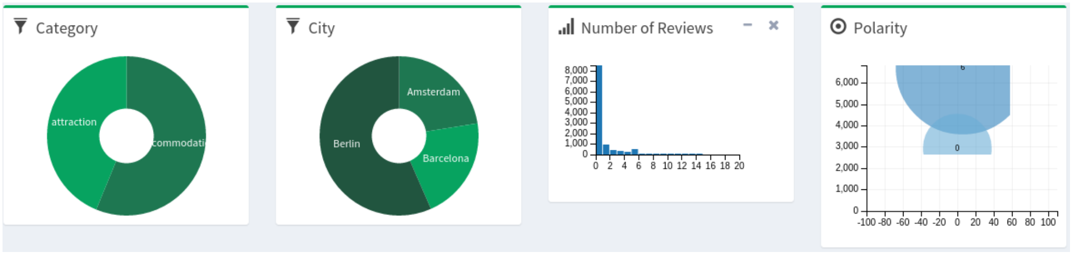
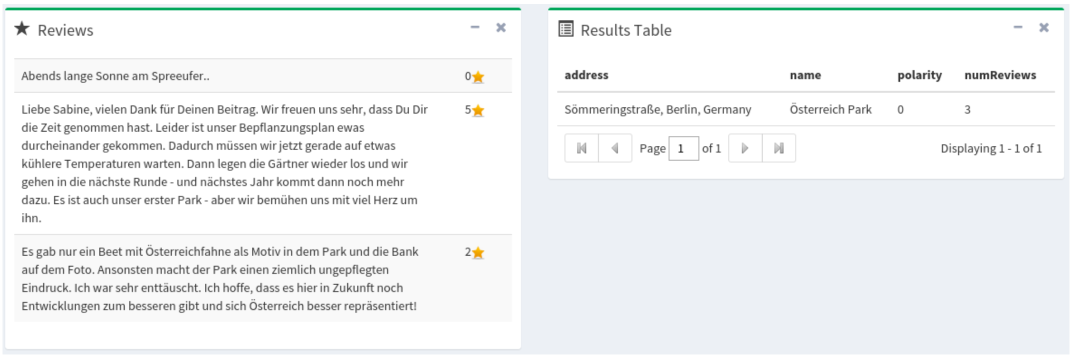


Figure 4.10: Filters

At last, we have implemented a custom widget, the Reviews widget, explained in detail in the ”Widgets” chapter; and the results table. The Reviews widgets will show reviews only in the case of selection of only one element:


Reviews

Abends lange Sonne am Spreeufer..	0★
Liebe Sabine, vielen Dank für Deinen Beitrag. Wir freuen uns sehr, dass Du Dir die Zeit genommen hast. Leider ist unser Bepflanzungsplan ewas durcheinander gekommen. Dadurch müssen wir jetzt gerade auf etwas kühlere Temperaturen warten. Dann legen die Gärtner wieder los und wir gehen in die nächste Runde - und nächstes Jahr kommt dann noch mehr dazu. Es ist auch unser erster Park - aber wir bemühen uns mit viel Herz um ihn.	5★
Es gab nur ein Beet mit Österreichfahne als Motiv in dem Park und die Bank auf dem Foto. Ansonsten macht der Park einen ziemlich ungepflegten Eindruck. Ich war sehr enttäuscht. Ich hoffe, dass es hier in Zukunft noch Entwicklungen zum besseren gibt und sich Österreich besser repräsentiert!	2★

Results Table

address	name	polarity	numReviews
Sömmeringstraße, Berlin, Germany	Österreich Park	0	3

Figure 4.11: Reviews and Table widgets

4.4.4 Conclusions

This dataset is full of information, with a rich set of facets for each feature and has a great number of features. We have clamp the total number of available features, taking only a certain number of places from 3 cities, but in the complete dataset we can find places from 8 cities, reaching more than 100.000 features.

Due to these facts, we accept it as a final demonstration dataset. A possible future work over this test would be storing the complete data on the server side and serve only the relevant data to the Sefarad 2.0 application, so the number of possible features being analysed would rise even more.

4.5 Summary

In this chapter we have described the three demonstrations that we have developed for Sefarad 2.0, detailing in each one their lacks and their advantages.

We have develop for each one a description of the data origins and the structure of this data as the pre-processing work that takes place. After that, we have justified the design of the widgets used for analysing this data.

CHAPTER 5

Evaluation

In this section we will present, test and analyse each prototype that we have developed for this project. We will list for each one the introduced features and will measure the quality of the filter system when the change is relevant.

5.1 Introduction

This section describes each stage the project has gone through, explaining the pros and cons of each step and why did we make each design decision.

We'll start with the last version that output the last project at our development group, Sefarad 1.0. In this project we produced a demo over a Slovakian dataset containing polygons and different data for each one. This dataset will be maintained through the different prototypes in order to obtain comparable and reliable conclusions.

Each section on this chapter describes one prototype, presenting images of the main features, a changes log and description and a benchmark.

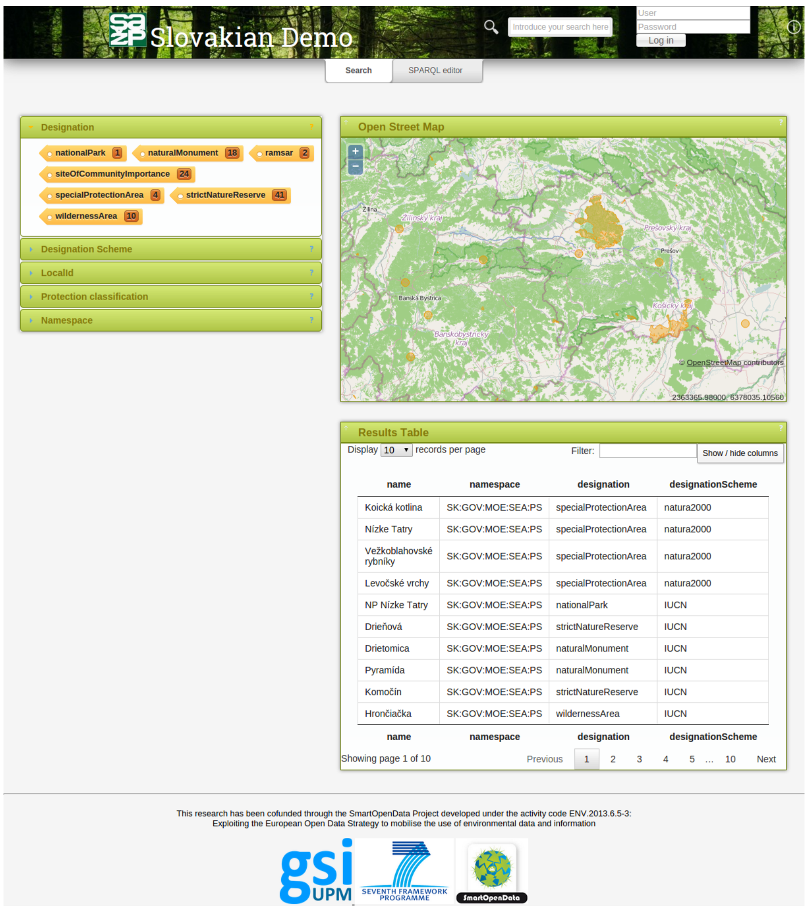
The benchmark is applied equally to all prototypes, being the "custom filter prototype" an exception. It consist on 5 different items:

- Page load time: time elapsed from the browser GET command to the data query sent action, when all sub-technologies are ready and waiting for the query data.
- Data retrieved time querying 100 elements: time elapsed from the query sent action to the data received and processed event (when all widgets have the data ready to read from).
- Data retrieved time querying 1000 elements: the same test with an extra stress to the loader system.
- Filter time for 100 data elements: time elapsed from the user click event to the point where the filtered data is ready.
- Filter time for 1000 data elements: the same test with an extra stress to the filter system.

5.2 Sefarad 1.0

This is our starting point. It has been developed through various projects and has grown with new features and expansions with each iteration.

5.2.1 Description

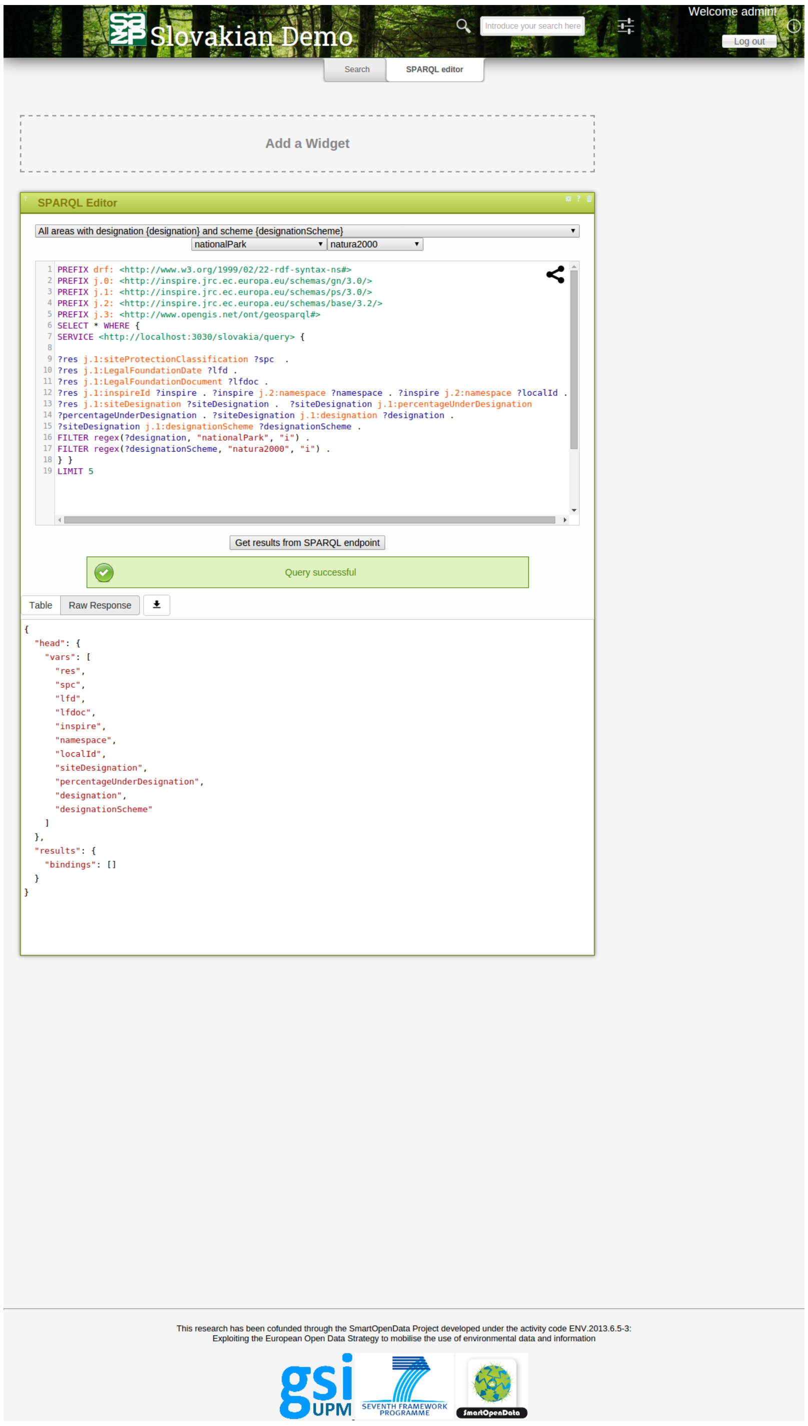


name	namespace	designation	designationScheme
Koická kotlina	SK:GOV:MOE:SEA:PS	specialProtectionArea	natura2000
Nízke Tatry	SK:GOV:MOE:SEA:PS	specialProtectionArea	natura2000
Vežkoblahovské rybníky	SK:GOV:MOE:SEA:PS	specialProtectionArea	natura2000
Levočské vrchy	SK:GOV:MOE:SEA:PS	specialProtectionArea	natura2000
NP Nízke Tatry	SK:GOV:MOE:SEA:PS	nationalPark	IUCN
Drieňová	SK:GOV:MOE:SEA:PS	strictNatureReserve	IUCN
Drietomica	SK:GOV:MOE:SEA:PS	naturalMonument	IUCN
Pyramída	SK:GOV:MOE:SEA:PS	naturalMonument	IUCN
Komočín	SK:GOV:MOE:SEA:PS	strictNatureReserve	IUCN
Hrončiačka	SK:GOV:MOE:SEA:PS	wildernessArea	IUCN
name	namespace	designation	designationScheme

Showing page 1 of 10 Previous 1 2 3 4 5 … 10 Next

This research has been cofunded through the SmartOpenData Project developed under the activity code ENV.2013.6.5-3: Exploiting the European Open Data Strategy to mobilise the use of environmental data and information

Figure 5.1: Sefarad-1.0 Dashboard



```
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.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 .
FILTER regex(?designation, "nationalPark", "i") .
FILTER regex(?designationScheme, "natura2000", "i") .
} }
LIMIT 5
```

```
{
  "head": {
    "vars": [
      "res",
      "spc",
      "lfd",
      "lfdoc",
      "inspire",
      "namespace",
      "localId",
      "siteDesignation",
      "percentageUnderDesignation",
      "designation",
      "designationScheme"
    ]
  },
  "results": {
    "bindings": []
  }
}
```

This research has been cofunded through the SmartOpenData Project developed under the activity code ENV.2013.6.5-3:
Exploiting the European Open Data Strategy to mobilise the use of environmental data and information

Figure 5.2: Sefarad-1.0 Query Editor

This version features various functional widgets:

- A map widget that is navigable and produces pop-ups on elements click, but is not capable of interaction with other widgets further than being updated when data is filtered (but it can not filter by itself).

- A data table, that includes a bunch of useful features like search and columns selection. As the rest of the widgets, it is capable of update its content each time the data is filtered.
- A couple of tag cloud widgets, organized in an accordion layout. This widget is the only one capable of actively filter the data by different facet values.
- A SPARQL editor widget included in a separated tab. It can hold multiple query templates with its respective values, that automatically updates the text editor bellow them. It can also execute the query and show the result in a table or raw form.

It also includes a custom filter system that works with the output of one query and process it, passing the result to each widget.

5.2.2 Benchmark

Test	**msec**
Page load time	395
Data retrieved time (100)	6625
Data retrieved time (1000)	40142
Filter time (100)	1312
Filter time (1000)	9344

5.2.3 Analysis

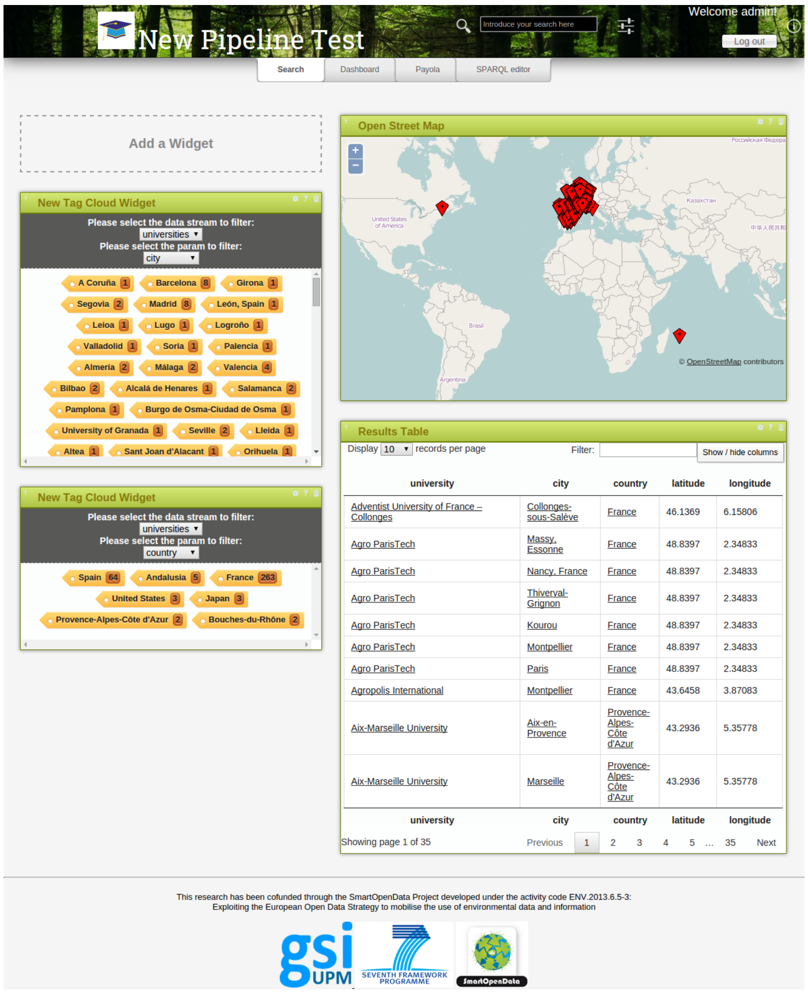
Although this version has served well for a number of implementations, it presents some remarkable problems:

- Custom filter System: The filter system malfunctions and is not optimised. It is coded inside the facet search widget's code, what means that no other widget could filter within this architecture.
- Poorly encapsulated code: Although there is a widget template definition, not all widgets implement it, so there are many widget behaviours coded directly in a main app file, which present difficulties to developers who might want to expand or modify the system.
- Non flexible css style: the style of the application is a mess, distributed in various files without a clear criteria and hardcoded.
- Not possible to add various widgets of the same type.
- Not scalable over hundreds of data elements.
- All widgets must share the same data, and is impossible to render in one widget the result of distinct queries (which could be useful to represent different layers in the map, for example).

5.3 Custom multi-query prototype

This prototype is an attempt to develop a brand new filter system which covers the main issues of the original sefarad.

5.3.1 Description



New Tag Cloud Widget

Please select the data stream to filter: universities

Please select the param to filter: city

A Coruña 1 · Barcelona 8 · Girona 1 · Segovia 2 · Madrid 8 · León, Spain 1 · Leioa 1 · Lugo 1 · Logroño 1 · Valladolid 1 · Soria 1 · Palencia 1 · Almeria 2 · Málaga 2 · Valencia 4 · Bilbao 2 · Alcalá de Henares 1 · Salamanca 2 · Pamplona 1 · Burgo de Osma-Ciudad de Osma 1 · University of Granada 1 · Seville 2 · Lleida 1 · Altea 1 · Sant Joan d'Alacant 1 · Orihuela 1

New Tag Cloud Widget

Please select the data stream to filter: universities

Please select the param to filter: country

Spain 64 · Andalusia 5 · France 263 · United States 3 · Japan 3 · Provence-Alpes-Côte d'Azur 2 · Bouches-du-Rhône 2

Display 10 records per page   Filter:

university	city	country	latitude	longitude
Adventist University of France – Collonges	Collonges-sous-Salève	France	46.1369	6.15806
Agro ParisTech	Massy, Essonne	France	48.8397	2.34833
Agro ParisTech	Nancy, France	France	48.8397	2.34833
Agro ParisTech	Thiverval-Grignon	France	48.8397	2.34833
Agro ParisTech	Kourou	France	48.8397	2.34833
Agro ParisTech	Montpellier	France	48.8397	2.34833
Agro ParisTech	Paris	France	48.8397	2.34833
Agropolis International	Montpellier	France	43.6458	3.87083
Aix-Marseille University	Aix-en-Provence	Provence-Alpes-Côte d'Azur	43.2936	5.35778
Aix-Marseille University	Marseille	Provence-Alpes-Côte d'Azur	43.2936	5.35778
university	city	country	latitude	longitude

Showing page 1 of 35   Previous 1 2 3 4 5 … 35 Next

This research has been cofunded through the SmartOpenData Project developed under the activity code ENV.2013.6.5-3: Exploiting the European Open Data Strategy to mobilise the use of environmental data and information

Figure 5.3: Custom multi-query prototype dashboard

It features multiple queries execution and filtering. It also allows multiple widgets of the same type. It is tested over a universities dataset, that in terms of load is similar to the Slovakian polygons used in Sefarad 1.0.

The new selectors div on top of the widgets allow users to select which dataset you want to render and filter on each widget.Abdullah Al

It worths the effort to specific how the filter system works in detail, since it marked the path to follow on the next prototyping iterations. The main architecture is the following:

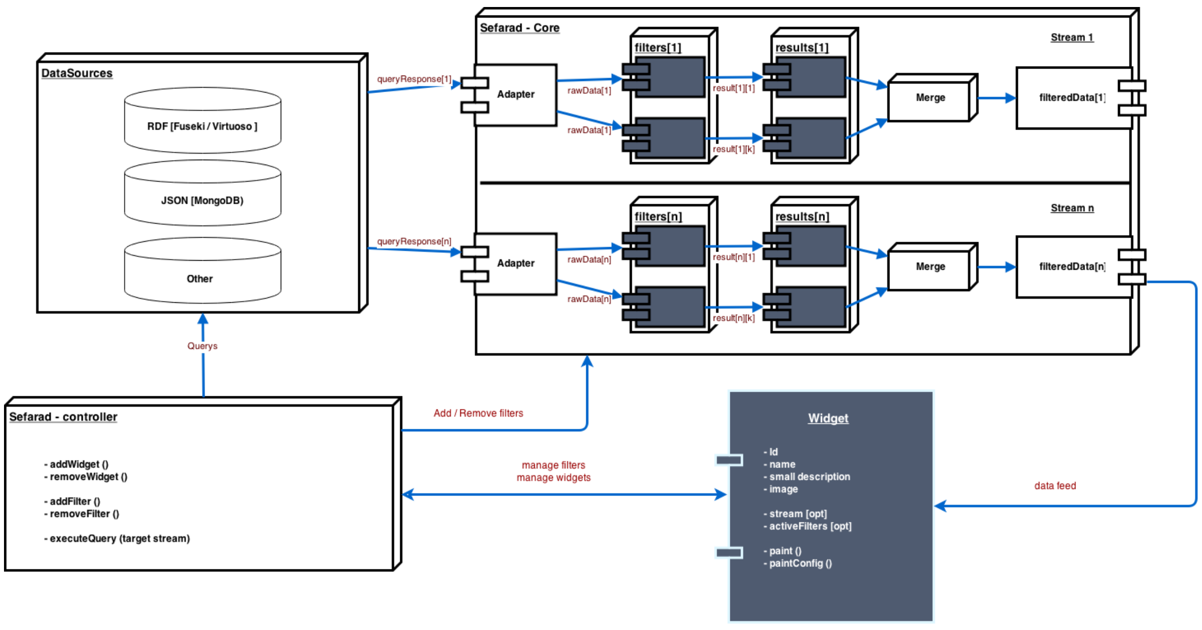


Figure 5.4: Multi Query Architecture

It aims to be a modular approach, so only one module of code would notice about the filter mechanism and incoming data, being each widget a mere filter productor and data receiver.

The controller is the module in charge of manage all widgets created. It can create, destroy, look for and update widgets. It also manages all request a widget might have, acting as interface between the widgets and the core.

The core implements various “streams” of data, acting like a parallel pipeline. It takes the incoming data from the online data sources and pass them iteratively through all filters of the stream. The output is still a set of independent filter results, so the core implements an array unique merge function that outputs the final response.

This filtered data is the income of the widgets, which can analyse this data and produce filters based on the user actions.

5.3.2 Benchmark

Test	msec
Page load time	393
Data retrieved time (342)	1316
Filter time (342)	8

5.3.3 Analysis

The prototype still doesn't solve many of the issues of the previous work like theme. It could be observed that the results of the benchmark say this is a fast algorithm, but we cannot take that as conclusion because we are working in different datasets.

Although this prototype works well, it also doesn't solve effectively the subtle mess code problem and still lacks a clear structure on some areas. Yet the filter between widgets is not implemented due the necessity of redo all widgets to match the new approach.

The prototype is still raw and we would need a couple more iterations to make it fully functional. However we might focus on the main structural problem and reformulate the entire code for future work, so this branch of the project has been left.

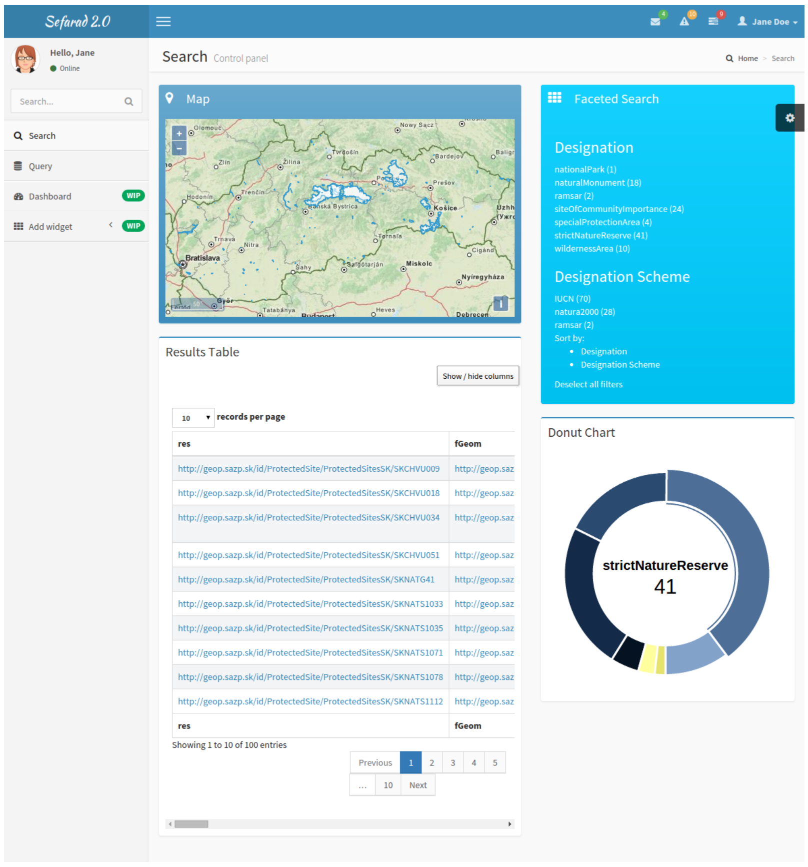
5.4 Bootstrap prototype

The next step in our development was cleaning the user interface, providing a flexible and well documented graphic framework to overcome the past problems.

We chose bootstrap to power these aspect of the app and, concretely, we based our appearance on the Admin-LTE free theme[1], property of Abdullah Almsaeed.

With this change, we eliminated the deep and messy code structure beyond the Sefarad 1.0 and included in this second version as less code as possible in order to keep it easy and clear.

5.4.1 Description



Designation
nationalPark (1)
naturalMonument (18)
ramsar (2)
siteOfCommunityImportance (24)
specialProtectionArea (4)
strictNatureReserve (41)
wildernessArea (10)

Designation Scheme
IUCN (70)
natura2000 (28)
ramsar (2)
Sort by:
- Designation
- Designation Scheme

Deselect all filters

Results Table

10 records per page

res	fGeom
http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK/SKCHVU009	http://geop.saz
http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK/SKCHVU018	http://geop.saz
http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK/SKCHVU034	http://geop.saz
http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK/SKCHVU051	http://geop.saz
http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK/SKNATG41	http://geop.saz
http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK/SKNATS1033	http://geop.saz
http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK/SKNATS1035	http://geop.saz
http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK/SKNATS1071	http://geop.saz
http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK/SKNATS1078	http://geop.saz
http://geop.sazp.sk/id/ProtectedSite/ProtectedSitesSK/SKNATS1112	http://geop.saz
res	fGeom

Showing 1 to 10 of 100 entries

Figure 5.5: Bootstrap Prototype Dashboard

[1]https://almsaeedstudio.com/preview

Under the graphical interface we have four widgets hardcoded in HTML5 and javascript, without a defined common structure yet:

- Datatable : similar to the one you could find in Sefarad 1.0. Some work was needed to adapt it to work under bootstrap. This datatable updates on data changes.
- OpenLayers map : the map from Sefarad 1.0 was developed under OpenLayers 1.0, that was not compatible with bootstrap. This was not a straight forward task, due to the differences between both versions in the projection procedures. The map updates on data changes, but doesn't offer further interaction.
- A donut chart taken from the resources of this adminLTE theme in order to demostrate that they could be easily integrated.
- A new filter system, very smart coded and modular. It's code creates the div containing the values selector and manages user interaction. It also feeds the rest of the widgets.

The architecture is much simple than its predecessors since it is the first prototype after a code cleaning and doesn't include much of the old features yet.

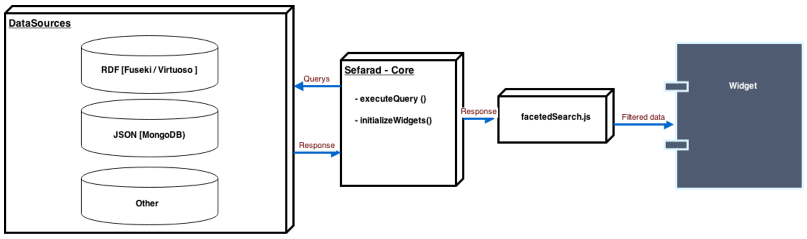


Figure 5.6: Bootstrap Prototype Architecture

5.4.2 Benchmark

Test	msec
Page load time	405
Data retrieved time (100)	5051
Data retrieved time (1000)	14442
Filter time (100)	779
Filter time (1000)	5047

5.4.3 Analysis

The benchmark does not show a qualitative improvement from Sefarad 1.0 in the speed aspect, but it improves indeed. The code is small and offers a solid ground to grow on.

FacetedSearch.js is a great library for filtering but sooner or later we would have to break in its code and redesign its modularity. At it's current state (jQuery plugin) is not possible to access some important aspect from the outside.

Bootstrap covers all our needs in the GUI area. We will stick with it for the next iterations.

5.5 Crossfilter.js + dc.js prototype

At this certain point of the development we discovered these two data visualization libraries, and after taking conscience of their specifications we inmediately started working on their integration, replacing facetedSearch and the current charts.

5.5.1 Description

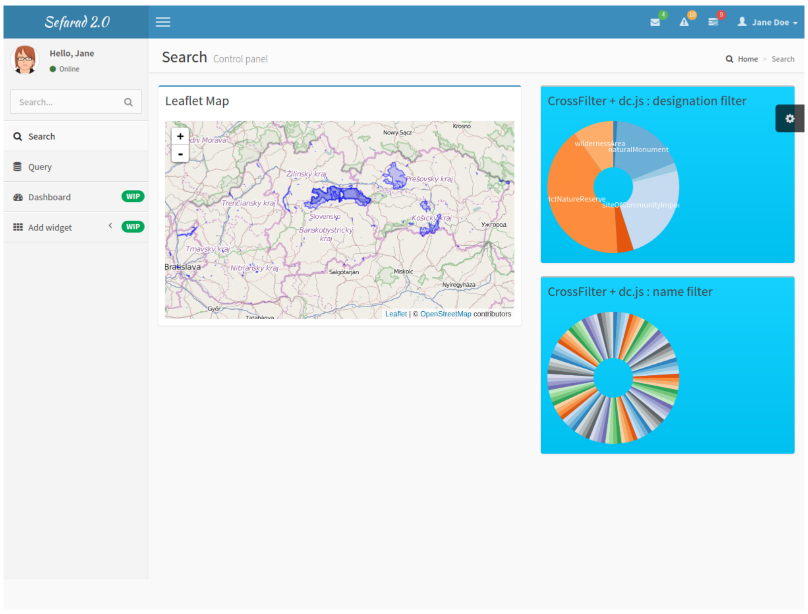


Figure 5.7: Crossfilter.js + dc.js dashboard

Crossfilter is a powerful library that handles thousands of data with ease and proves very helpful in the process of defining and grouping different data dimensions.

Dc.js is a beautiful chart renderer that hides in its insides a very well constructed system that handles the filters and events of all the charts for us. From now on, each widget in Sefarad that needs to filter or get filtered data will implement this chart structure inside. For the moment we have not defined a clear widget structure, but this characteristic will be crucial at the time of choosing it.

We have also changed the OpenLayers module for leaflet, a much more elegant and user friendly solution that can we integrated inside a dc chart. In fact, every graphic library can be framed inside a base dc chart, but is not an easy task, as dc lacks documentation. Luckily, we found the previous work of Boyan Yurukov[2], Bulgarian programmer, accomplishing this same task, so we followed his steps and learn from it for future includes.

As result, the prototype features an interactive map, which can show a pop-up with the name of the area and select it (having an impact on the rest of the widgets) and two donnut charts for demonstration purposes.

5.5.2 Benchmark

Test	**msec**
Page load time	160
Data retrieved time (100)	5598
Data retrieved time (1000)	14582
Filter time (100)	5
Filter time (1000)	27

5.5.3 Analysis

It takes practically the same time to retrieve the data elements than the same case in Sefarad 1.0 but it drastically improves the performance on filtering.

After the benchmarks we decide to continue with crossfilter and dc due to their speed and smartness.

[2]https://github.com/yurukov/dc.leaflet.js

5.6 Final Prototype: Web Components

5.6.1 Description

After the choice of the filter system, we just needed a widget structure. Until this point, all widgets developed for Sefarad 2.0 where chunks of HTML and code exposed at the main level. We started looking for a system that encapsulates each widget, so we can focus the development on them without alter the rest of the framework.

This is where we include the web components philosophy in Sefarad 2.0, and we chose Polymer as a framework to develop components.

5.6.2 Benchmark

We have maintained crossfilter as our filter system and the dataset chosen (Slovak data), so the metrics are nearly the same of that case, with the web components not having a noticeable impact on the filter time, but having a slight impact on the page load time(around 100 msec). This delay is produced by web components initialization, where they have to inject their HTML templates and execute their init function. However, we evaluate this time descent as acceptable given the advantages that web components offers to our application.

Test	**msec**
Page load time	240
Data retrieved time (100)	5634
Data retrieved time (1000)	14653
Filter time (100)	5
Filter time (1000)	27

5.6.3 Analysis

We have reached a point on our development process where we got an stable and reliable filter system and a convincing widget structure, so we can start growing from this point to build some dashboards and face case study challenges.

The complete architecture and the working details are full explained in the chapter "Sefarad 2.0", as this prototype has become our actual application and has grown with a lot of widgets and extra functionality.

5.7 Prototype comparison

Test	Sefarad-1.0	Bootstrap	Crossfilter + dc	Web Components
Page load time	395	405	160	240
Data retrieved time (100)	6625	5051	5598	5634
Data retrieved time (1000)	40142	14442	14582	14653
Filter time (100)	1312	779	5	5
Filter time (1000)	9344	5047	27	27

The multi-query prototype is left out of this comparison due to its use of a different dataset.

5.8 Summary

In this chapter we have reviewed every development stage of the project, from the predecessor, Sefarad 1.0, to the current prototype.

We have set a few metrics to measure for each one in order to compare them and justify our development decisions. For each one, we have taken that measures and given conclusions about advantages and disadvantages of each technology or integration, as well as notes for the next iteration.

As result, we have a set of technologies with a justified election that we are going to use in Sefarad 2.0.

CHAPTER 6

Conclusions and future lines

This chapter summarizes the conclusions extracted from this master thesis and evaluates if the objectives were achieved. After that, we discuss thoughts about future work.

6.1 Achieved goals

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

Study and test different web technologies reaching conclusions for each one under certain criteria. This goal has been achieved successfully. Its results are presented in Chapter 2, where we study and analyse the different technologies considered for the different facets of the project .

Design the Architecture of the application through prototype iteration. This goal has been achieved successfully. The complete architecture of the system and a detailed explanation of all its modules and sub-modules is included in Chapter 3. Furthermore, this iterative process is described in great detail in Chapter 5.

Compare and document the features of each iteration, benchmarking where possible. This documentation and measurement has been also achieved, being the entire task report located in Chapter 5.

Develop one or more case studies to test the final application and demonstrate its possibilities. The result of this challenge has been evaluated in Chapter 4. We have gone through three different use cases, analysing each one and evaluating their pros and cons until we have reach a successful stage.

Document the final application to ease future developments or use cases. We have committed the Chapter 3 to the detailed explanation and documentation of all features of Sefarad 2.0, ending up with a well structured reference material for future Sefarad developers.

6.2 Conclusions

This project has allowed us to perform a deep insight into the *Semantic Web* and *Linked Data* and all its benefits. We have developed a functional application capable of query, sort and filter linked data.

Part of this project has been developed in the scope of the SmartOpenData project contributing to this European project, so we have worked with companies such as Tragsa[1]. To work in a working group has helped us to organize tasks and responsibilities and has forced us to organize with our partners.

We have used existing advanced technologies whenever it was possible, studying in depth web components philosophy, dart workflow and dc.js integration issues. We have put all of them together to build a solid and functional system. We have left a lot of tools and frameworks on the way, not in vane but learning from each one to build a new and more solid project stage.

We have learn from past experiences developing for Sefarad 2.0, identifying its weaknesses and designing a new architecture that assures the same functionality with easier development.

We experienced big changes as early technology adopters, such as new versions of the dart language and Polymer framework, 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.

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

6.3 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, as well as areas of possible direct application of our framework:

- Develop an installer for Sefarad 2.0.
- Add widgets on the fly, developing a new set of graphical tools for the selection and parameters settings.
- Explore the natural language processing area, implementing new widgets and core functionalities that able a non technical user to query and explore the semantic web.
- Apply Sefarad 2.0 to new projects where data graphical analysis is needed, developing new custom widgets in order to face new problems.
- Save dashboards with their complete widget configuration in our mongoDB repository.

Sefarad 2.0 is still young. As developers, we can find a lot of new functionality examples in older web and tools as Payola, lodLive [2], cartoDB [3], etc.

All new ideas we could come across can be implemented in this new framework thanks to the modularity of web components and the power of the rest of our technology selection.

[2]http://en.lodlive.it/
[3]http://cartodb.com/

APPENDIX A

How to build a dashboard

In this appendix we will explain the process we have followed to construct the demo dashboards that can be found on the demo of Sefarad 2.0.

A.1 Design

First come the information analysis: which fields are the results going to have? how many? which type?

Once we know this information, we have to decide which widgets we want to integrate in the dashboard and which facets is going to render each one.

Now that we have the information analysis design, its time to put it on the board. First we have to create a new empty dashboard copying an existing one and erasing all its components. Now we have to import all widgets that we are going to use in the ”Import Zone” (A.1).

```
<!-- POLYMER COMPONENTS IMPORT ZONE -->
    <link rel="import" href="polymer-elements/pie-chart/pie-chart.
        html">
    <link rel="import" href="polymer-elements/bar-chart/bar-chart.
        html">
    <link rel="import" href="polymer-elements/number-chart/number-
        chart.html">
    <link rel="import" href="polymer-elements/leaflet-map/leaflet-
        map.html">

    <link rel="import" href="bower_components/sortable-table/
        sortable-table.html">
    <link rel="stylesheet" href="bower_components/sortable-table/
        css/bootstrap.css" shim-shadowdom>
    <link rel="import" href="polymer-elements/faceted-search/
        faceted-search.html">
    <link rel="import" href="polymer-elements/paper-shadow/paper-
        shadow.html">
    <link rel="import" href="polymer-elements/paper-tab/paper-tabs.
        html">
    <!-- end/ POLYMER COMPONENTS IMPORT ZONE -->
```

Listing A.1: Web Components Import Zone

Now we have to place the custom tags of the widgets with their corresponding parameters. See the widgets section of this work for reference. The placement can be done as the user wants, but we encourage the user to follow the AdminLTE template, wrapping the actual Polymer element inside the proper bootstrap divs, so we doesn't break the current look and feel of Sefarad 2.0. But, out of this recommendation, the user has total liberty of placement for his components.

A.2 Widget Definition Example

Lets give an example of placing a widget in the dashboard, applied to the restaurants dataset:

We start with an empty dashboard, as shown in figure A.1.

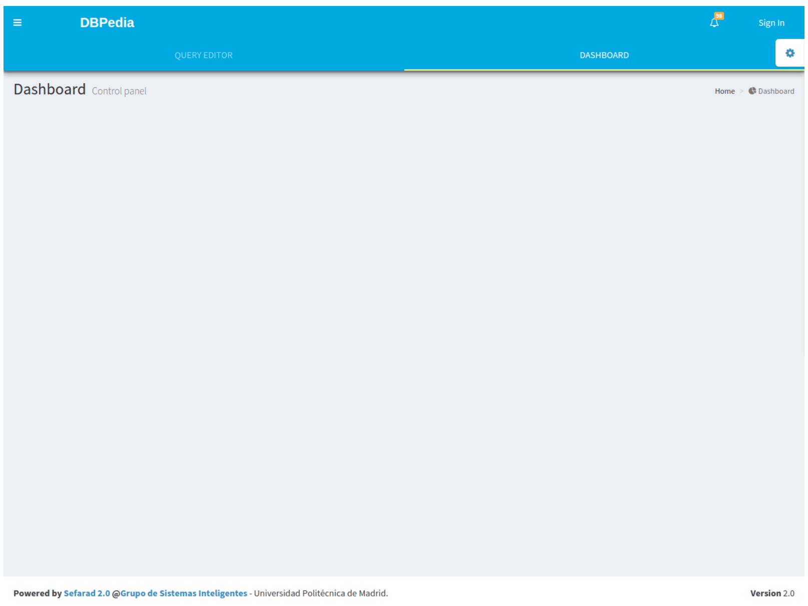

DBPedia
Sign In
QUERY EDITOR
DASHBOARD
Dashboard Control panel
Home > Dashboard
Powered by Sefarad 2.0 @Grupo de Sistemas Inteligentes - Universidad Politécnica de Madrid.
Version 2.0

Figure A.1: Navigation Menu

Then we decide to include a pie chart widget that shows the percentage of restaurants that are able to handle reservations. Then we have to create a bootstrap box and instantiate our pie chart web component inside (A.2), taking care of setting its parameter to "newReservations", the facet we want to analyse.

```
<div class="col-lg-3 col-xs-6">
    <!-- pie-chart -->
    <!--Widgets must have 'widget' class in order to render their
        loading screen-->
    <div class="box box-primary widget">
        <div class="box-header">
            <i class="fa fa-inbox"></i>
            <h3 class="box-title" id="here">Handle Reservations</h3
                >
        </div>
        <!-- box-body -->
        <div class="box-body chart-responsive">
            <pie-chart param="newReservations" class="dc-element"
                ></pie-chart>
        </div>
        <!-- /.box-body -->
    </div>
    <!-- /.pie-chart -->
</div>
```

Listing A.2: Widget Insertion

The result of this is the dashboard shown in figure A.2.

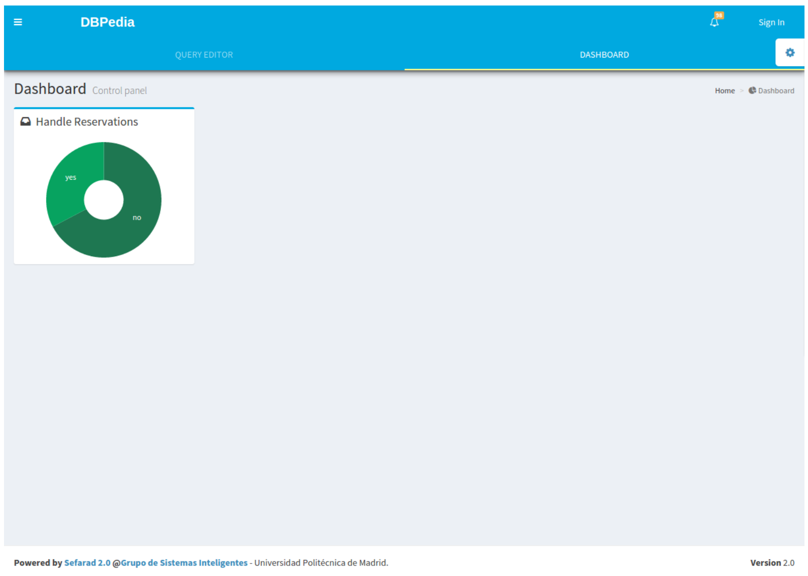


Figure A.2: Navigation Menu

A.3 Pre-processor

Once the widgets are ready, we have to write a preprocessor for this dashboard. In the body div we have set an identifier id that we can retrieve form the controller, so we apply one branch of pre-process code or another. In the future, this will be replaced by Dart code. In the current state of the project however, this is how it is done.

For an example of this is achieved, please go to the ”preprocessor architecture” section.

Once the preprocessor is written, all will be set for feeding the widgets with data.

Now all that is left to do is reference this new dashboard in the rest of the web pages of Sefarad 2.0, writing the corresponding code for the left navigation menu section:

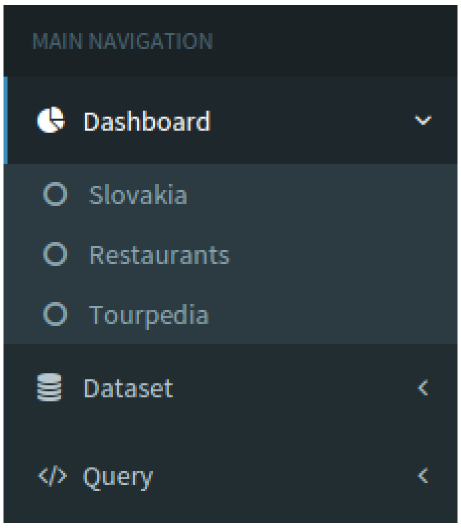


Figure A.3: Navigation Menu

Bibliography

[1] T. Berners-Lee, Hendler, *et al.*, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 28–37, 2001.

[2] F. J. Lopez-Pellicer, M. J. Silva, M. Chaves, F. J. Zarazaga-Soria, and P. R. Muro-Medrano, "Geo linked data," in *Database and Expert Systems Applications*, pp. 495–502, Springer, 2010.

[3] O. Lassila and R. R. Swick, "Resource description framework (rdf) model and syntax specification," 1999.

[4] C. Steenmans, M. Vanderhaegen, H. De Groof, J. Martin, I. Livbjerg, J. Ryttersgaard, J. Sievers, M. Reuvers, A. Lillethun, A. Linsenbarth, *et al.*, "Inspire scoping paper," 2004.

[5] P. Archer, K. Charvat, M. Navarro, C. A. Iglesias, J. O'Flaherty, T. Robles, and D. Roman, "Linked open data for environment protection in smart regions–the smartopendata project," 2013.

[6] O. K. Ondra Heřmánek and Others, "Payola," 2014.

[7] R. Team, "Redash.io," 2015.

[8] D. Cooney, "Introduction to web components," tech. rep., W3C, july 2014. Available at `http://www.w3.org/TR/components-intro/`.

[9] I. Square, "crossfilter," 2015.

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

[11] 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.

[12] 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.

[13] 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.

[14] R. Díaz-Vega, "Design and implementation of an HTML5 Framework for biodiversity and environmental information visualization based on Geo Linked Data," Master's thesis, Universidad Politécnica de Madrid, ETSI Telecomunicación, December 2014.

[15] S. Skelston, “sortable-table github,” 2015.