TRABAJO DE FIN DE GRADO

Título:	Desarrollo de un Sistema de Monitorización de Medios Sociales basado en Elasticsearch y Tecnologías de Componentes Web
Título (inglés):	Development of a Social Media Monitoring System based on Elasticsearch and Web Components Technologies
Autor:	Enrique Conde Sánchez
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	Mercedes Garijo Ayestarán
Vocal:	Álvaro Carrera Barroso
Secretario:	Juan Fernando Sánchez Rada
Suplente:	Tomás Robles Valladares

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

TRABAJO DE FIN DE GRADO

DEVELOPMENT OF A SOCIAL MEDIA MONITORING SYSTEM BASED ON ELASTICSEARCH AND WEB COMPONENTS TECHNOLOGIES

Enrique Conde Sánchez

Junio de 2016

Resumen

Con el advenimiento de la web social, los usuarios expresan sus opiniones y comentarios en los medios sociales. Esta información generada por el usuario se ha convertido en un activo valioso para entender a la gente y sus opiniones y poder aprovechar los conocimientos obtenidos a partir de encuestas con una población objetivo. Sin embargo, para entender la sabiduría de la multitud, se necesita desarrollar tecnologías para recoger, filtrar, analizar y visualizar los medios de comunicación social. Las herramientas de monitorización de medios sociales proporcionarán estas comodidades.

Este proyecto fin de carrera tiene como objetivo desarrollar un sistema de monitorización de medios sociales basados en tecnologías semánticas. Las principales funcionalidades del sistema serán los siguientes: (i) Recoger información de las redes sociales utilizando rastreo y APIs disponibles; (ii) La capacidad para llevar a cabo el análisis semántico y exponer estas anotaciones como datos vinculados; (iii) Consulta semántica y filtrado; (iv) La visualización y búsqueda facetada; y (v) La programación de análisis.

Para este propósito, el sistema se beneficiará de los sistemas y servicios disponibles en el laboratorio GSI, como GSI Crawler o el servicio de análisis de sentimientos y emociones Senpy.

Las principales tecnologías utilizadas en el trabajo final serán ElasticSearch y las tecnologías de componentes web. ElasticSearch será utilizado para la indexación y consulta de los medios sociales de una manera escalable. En cuanto a los componentes web, serán utilizados para permitir la adaptación de la herramienta a diferentes casos de uso.

Se evaluará el sistema a través del desarrollo de varios casos de uso, en el que se evaluaron los aspectos de rendimiento y facilidad de uso.

Palabras clave: Tecnologías semánticas, RDF, SPARQL, Componentes Web, Polymer, JavaScript, Elasticsearch, GSI Crawler, Dashboard, Luigi

Abstract

With the advent of the social web, users express their opinions and comments in social media. This user generated information has become a valuable asset to understand people opinions and can leverage the insights obtained from surveys with a targeted population. Nevertheless, to understand the wisdom of the crowd, it is needed to develop technologies to collect, filter, analyse and visualise social media. Social media monitoring tools provide these facilities.

This final project aims at developing a social media monitoring system based on semantic technologies. The main functionalities of the system will be: (i) collecting social media using crawling and available APIs; (ii) ability to perform semantic analysis and expose these annotations as Linked Data; (iii) semantic query and filtering; (iv) visualisation and faceted search; and (v) scheduling of analysis.

For this purpose, the system will benefit from systems and services available in the GSI laboratory, such as GSI Crawler or Senpy Sentiment and Emotion Analysis services.

The main technologies used in the final work will be Elastic Search and Web Components. Elastic Search will be used for indexing and querying social media in a scalable way. Regarding Web Components, they will be used to enable the adaptation of the tool to different use cases.

The system will be evaluated through the development of several use cases, where performance and usability aspects will be evaluated.

Keywords: Semantic technologies, RDF, SPARQL, Web Components, Polymer, JavaScript, Elasticsearch, GSI Crawler, Dashboard, Luigi

Agradecimientos

Primero agradecer a mis padres y a mis hermanos, por todos estos años a mi lado, apoyandome, confiando en mi y ofreciendo todo lo que tenían para que yo cumpliera mi sueño.

Segundo, a mi novia, mi guía, mi luz, mis manos y mis ojos. Es el apoyo que toda persona necesita y reordenó mi vida hasta ser lo que soy ahora.

A mis abuelas, por el cariño que me han brindado durante toda mi vida, gran parte de lo que soy hoy es por ellas.

En especial, mención a mis abuelos y mi tío que he sentido su apoyo en todo momento, siempre he sentido que tenía un ángel de la guarda, pero tengo 3 ángeles, que estén donde estén, siempre estarán a mi lado y nunca les olvidaré.

A mi amiga Cristina, a mi lado desde que entramos en pre-escolar, los momentos difíciles se nota cuando un amigo es para siempre, y ella siempre estaba a mi lado, esto también te lo debo a tí.

A mis compañeros de laboratorio, por el ambiente de trabajo tan grato que se ha creado, por la relación que se establece aunque no nos conociéramos y las ayudas incondicionales que te ofrecen.

Y por último, y no menos importante, a Carlos Ángel, mi tutor, por la confianza depositada en mi, por la oportunidad única que me ofreció de formar parte del grupo y por ser el guía de mi proyecto y hasta ahora corta vida laboral. Gracias de corazón.

“Nuestra recompensa se encuentra en el esfuerzo y no en el resultado. Un esfuerzo total es una victoria completa”. **Mohandas Karamchand Gandhi**

Contents

List of Figures

CHAPTER 1

Introduction

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 analyzed data in the semantic web. This is the case of Financial Twitter Tracker [8], a R&D Spanish project that analyzes data from Twitter which are related to a company and analyze them storing the result, and it could be represented graphically with charts to get opinions, sentiments about that topic.

Tourpedia[12] 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 Tourpedia dataset.

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

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, we have created a new workflow attempting to provide a intuitive interface between users and datasets and offering some graphic tools to represent these data.

1.2 Project goals

The goal of this Project is to develop a web application that allows a non-technical user to query 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 main feature that puts Sefarad apart is its analysis dashboard, capable of aggregated search and render thousands of data thanks to the power of bleeding edge technologies as web components and ElasticSearch, 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.
- Develop one or more case studies to test the final application and demonstrate its possibilities.
- Design the architecture of the application through prototype iteration.
- Document the final application to ease future developments or use cases.

1.3 Structure of this Project

In this section we will provide a brief overview of all the chapters of this Project. 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 project 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: makes a requirement analysis which will enable a more complete vision of the system, listing the use cases of the system.

Chapter 4: describes in depth all the important aspect of Sefarad 3.0, our product, Defining its architecture, its interaction model.

Chapter 5: describes the two 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 6: 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 ElasticSearch, Luigi, and Web Components, the three most important tools of this project. Then we'll move on presenting all technologies that enable us to build a semantic web with a data analyzing, data filtering, and rendering system.

2.1 ElasticSearch

Scalability is the main problem which both first versions of *Sefarad* have faced when they were made. They had *Sparql* support but, after time when more data were included, page loading time was intolerable and it could not continue in this way.

Elasticsearch [3] is a search server based on Lucene. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents.

Elasticsearch is able to achieve fast search responses because, instead of searching the text directly, it searches an index instead.

This type of index is called an inverted index, because it inverts a page-centric data structure (page to words) to a keyword-centric data structure (word to pages). *Elasticsearch* uses Apache Lucene to create and manage this inverted index.

It includes some frameworks like *Logstash* [5] or *Kibana* [4] that make *ElasticSearch* very useful and good in a comprehensive way.

- *Logstash* is a tool for managing events and logs. When used generically the term encompases a larger system of log collection, processing, storage and searching activities. It enables collecting data via configurable input plugins, in this way, we use Twitter plugin to recover some tweets and after that *Logstash* process them by any number of filters which modify and annotate the event data. Finally it routes events to output plugins which can forward the events to *Elasticsearch*.

- *Kibana* is a browser-based analytics and search interface for *ElasticSearch* developed for viewing logstash events.In this way, we could create some charts, both normal and aggregates, and put them in Dashboards to analyze data.

2.2 Luigi

Facing with a string of tasks as recover data, send them to analyze and storing them in ElasticSearch could be difficult to handle. As a consequence, we look for some solutions for this problem, that it can be as easy as plug&play and compatible with ElasticSearch. *Luigi*[7] was the solution. It lets assign tasks one by one and handle problems that may arise. Besides this, a web interface could be used to see how tasks are managed and assign some other tasks to it. Using Luigi's visualizer, we get a nice visual overview of the dependency graph of the workflow. Each node represents a task which has to be run. Green tasks are already completed whereas yellow tasks are yet to be run. Luigi is similar to GNU Make where you have certain tasks and these tasks in turn may have dependencies on other tasks. There are also some similarities to Oozie and Azkaban. One major difference is that Luigi is not just built specifically for Hadoop, and it's easy to extend it with other kinds of tasks.

2.3 Web Components

Web Components[13] are a set of standards currently being produced by Google engineers as a W3C specification that enables 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 component 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, described below.

2.3.1 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.

2.3.2 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.

2.3.3 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.

2.3.4 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.

2.4 Polymer

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

Polymer makes 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.

Structure of Polymer elements:

Listing 2.1: Custmos element name-tag

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

<dom-module id='name-tag'>
  <template>
    <!-- bind to the 'owner' property -->
    This is <b>{{owner}}</b>'s name-tag element.
  </template>

  <script>
    Polymer({
        is: 'name-tag',
        ready: function() {
        // set this element's owner property
          this.owner = 'Enrique';
        }
    });
  </script>
</dom-module>
```

Listing 2.2: Usage of custom element name-tag

```
<!DOCTYPE html>
<html>
  <head>
    <script src='bower_components/webcomponentsjs/webcomponents-lite.min.js'></script>
    <link rel='import' href='name-tag.html'>
  </head>
  <body>
    <name-tag></name-tag>
  </body>
</html>
```

2.5 Semantic Technologies

Semantic Technologies 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. 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 (also called Web of Data).

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

2.5.1 RDF

Resource Description Framework (RDF)[1] 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:

Listing 2.3: RDF/XML document example

```
<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>
```

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.

[1]https://www.w3.org/RDF/

2.5.2 SPARQL

Linked Data can be queried using SPARQL[10] (an acronym for SPARQL Protocol and RDF Query Language), a query language for RDF which became an official W3C Recommendation[2]. 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:

Listing 2.4: SPARQL query example

```
prefix sch-ont:   <http://education.data.gov.uk/def/school/>

select ?name WHERE {
  ?school a sch-ont:School.
  ?school sch-ont:establishmentName ?name.
  ?school sch-ont:districtAdministrative <http://statistics.data.gov.uk/id/local-authority-
       district/00AA>.
}

order by ?name
```

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.

2.5.3 Fuseki

Fuseki[3] is a SPARQL server. It can run as a service, as a Java web application, and as a standalone server. It provides security (using Apache Shiro) and has a user interface for server monitoring and administration.

It provides the SPARQL 1.1 protocols for query and update as well as the SPARQL Graph Store protocol.

Fuseki is tightly integrated with TDB to provide a robust, transactional persistent storage layer, and incorporates Jena text query and Jena spatial query. It can be used to provide the protocol engine for other RDF query and storage systems.

[2]http://www.w3.org/blog/SW/2008/01/15/
[3]https://jena.apache.org/documentation/serving_data/

2.6 GSI Crawler

GSICrawler [2], a service, with an online demo, that will collect and analyze the comments from the different websites. Implementation of a service to schedule, monitor and administrate the crawling system. It will be described the development of scrapers to collect comments. This website is useful to the analysis of comments from any social network or social website. The user will choose the type of analysis he wants to carry out (Emotions, Sentiments or Fake Analysis) and the user will also supply, for instance, a direct URL to a Yelp's Business, the id of a Facebook's Fan Page or a YouTube's Video. GSI Crawler will download the comments belonging to this element and, later, the pertinent analysis will be run using the Senpy tool. Once the analysis is finished, a summary of the result will be shown and the possibility of review each comment one by one will be also offered.

In GSI Crawler, there are *Spiders* that they work using HTML Parser. Data of the same category are typically encoded into similar pages by a common script or template. In data mining, a program that detects such templates in a particular information source, extracts its content and translates it into a relational form, is called a wrapper. Wrapper generation algorithms assume that input pages of a wrapper induction system conform to a common template and that they can be easily identified in terms of a URL common scheme.

2.7 Senpy

Senpy [9] is an open source reference implementation of a linked data model for sentiment and emotion analysis services based on the vocabularies NIF, Marl and Onyx.

A modular approach allows organizations to replace individual components with custom ones developed in-house. Furthermore, organizations can benefit from reusing prepackages modules that provide advanced functionalities, such as algorithms for sentiment and emotion analysis, linked data publication or emotion and sentiment mapping between different providers.

To sum up, it provides a service that inserting data, them will be analyzed through different algorithms with linked data, and emotion and sentiment analysis. These processed data will be send them back through REST API.

CHAPTER 3

Requirement Analysis

3.1 Introduction

The result of this chapter is a requirement analysis which will enable a more complete vision of the system to be developed. Besides, this chapter also helps the reader in the process of understanding the purpose of the Sefarad-3.0 project.

The analysis will use the Unified Modeling Language (UML)[1]. This language allows us to specify, build and document a software system using graphical language.

This analysis is important when understanding a system, but also when designing a software system. Because of this, we present this analysis chapter.

That being said, the aim of this chapter is not to cover all the system requirements, or all the Sefarad functionalities. In this chapter the analysis will be made briefly; it is not a thorough analysis of the Sefarad system.

[1]http://www.uml.org/

3.2 Use cases

This section identifies the main use cases of the Sefarad system. This helps to obtain the specifications of the uses of the system, and therefore defines a list of requisites to match.

In 3.2.1, a list of the main actors will be presented and a UML diagram representing all the actors participating in the different use cases in 3.2.2. This representation allows us to identify the actors that interact with the system, as well as the interconnection between them. Then, several subsections -3.2.2.1 and 3.2.2.2- will show the sequence diagrams of some of the use cases. The sequence diagrams are developed following the UML language.

3.2.1 System actors

Identifying the actors of the system is the first step to take into consideration when an analysis of a system is being made. The actors of the Sefarad system are:

User. Final user of the system, and the main actor. It accesses the Sefarad system aiming to make queries using the available offered queries or making its own queries to the default endpoint. This actor could also search data indexed into ElasticSearch, or simply manages filters used in the dashboard to retrieve data it wants to.

Admin. Administrator of Sefarad, in charge of managing dashboards, could create new dashboards, edit other dashboards or delete them. It is able to index new data in ElasticSearch service, manage filters in a different way than users, creating new patterns or allowing to filter in different fields. It could also create new workflows of Luigi service, and explore them in Luigi web service.

ElasticSearch. This is a secondary actor. It stores data that were indexed and new data that are sent and retrieve them through a JavaScript Library making a REST petition.

Fuseki. This is another secondary actor. It recieves queries from any dashboard and send back data to user in order to look into them. It only accepts Sparql language.

3.2.2 Use cases

Next a use case diagram is presented. In this graphic it is shown the main Sefarad use cases, and the interconnection with the actors of the system.

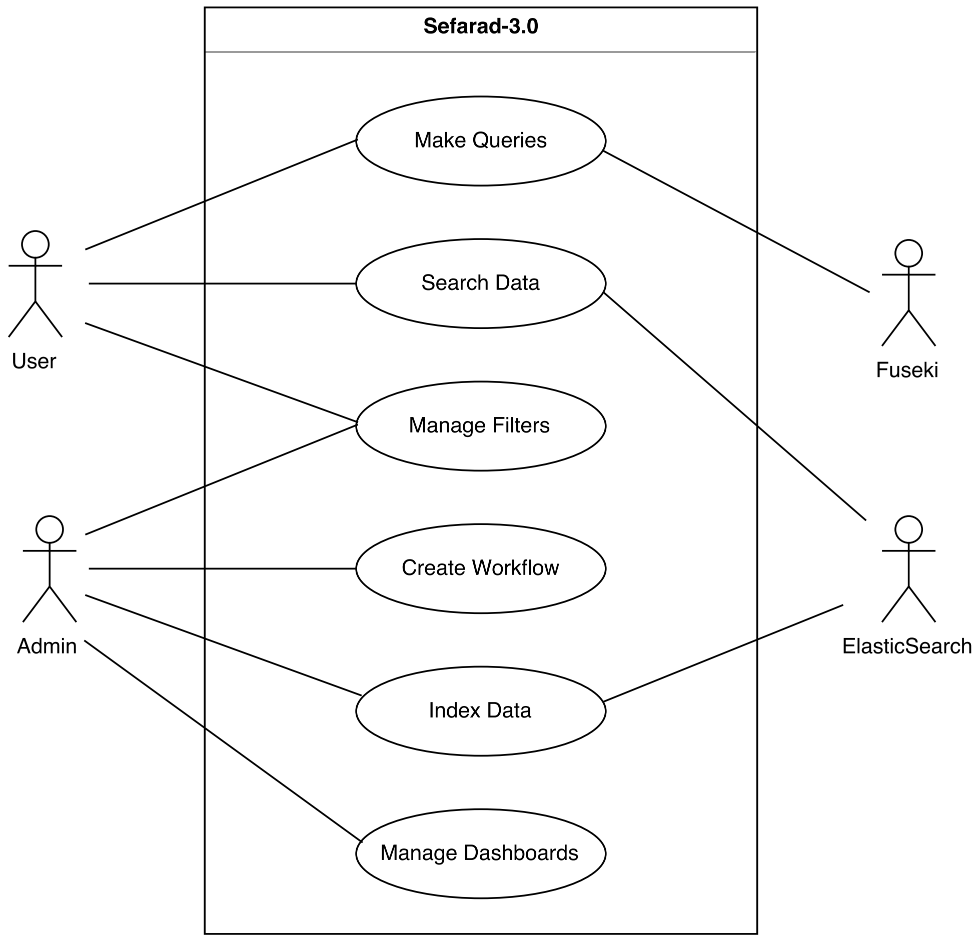


Figure 3.1: Use case diagram

3.2.2.1 Make queries

In this use case the main actor is the User and the secondary actor is Fuseki. This actor accesses any dashboard via web, and the it could execute default queries or write their own queries, following the process that can be seen in figure 3.2. Thus, results will be deployed in a fragment where you can filter data received using a pivot table, Google Chart, or simply explore data in a json or results table. It can be possible making a request from YASGUI library through a REST petition to the available Fuseki service on the server.

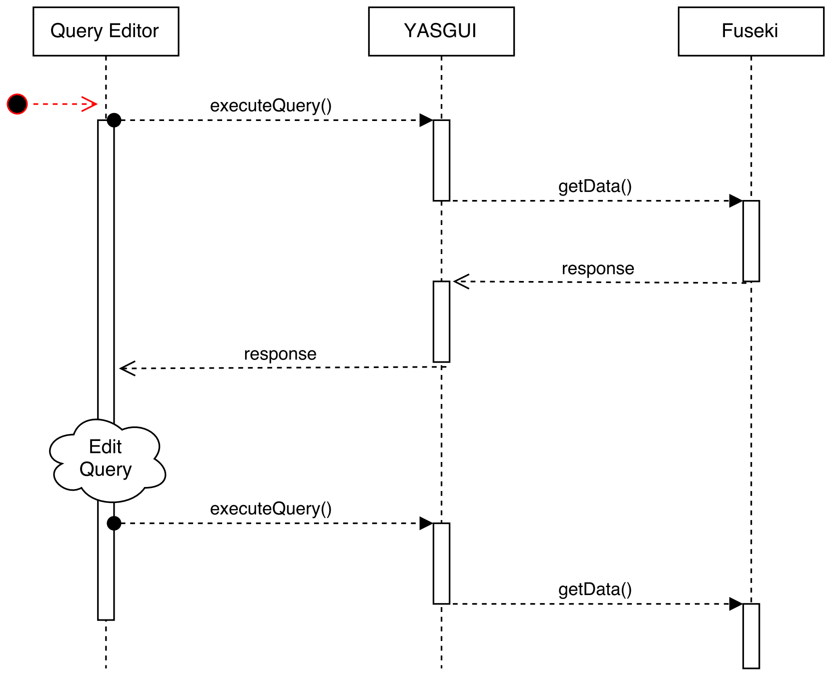


Figure 3.2: Make queries case use

3.2.2.2 Create Workflow

The actor of this use case is the Admin. In this case, the admin accesses to Luigi service and execute a workflow previously created. It must to be created in Python. A default workflow was created and it can be seen in figure 3.3. A task would be created and will call other tasks which are dependent of the first one. They will run command lines and will response with an output method, to serve those data to the next task. Once the workflow is completed, it will notice the user the result of the execution, if it was right or wrong. If the result was pending, the task is scheduled to run later.

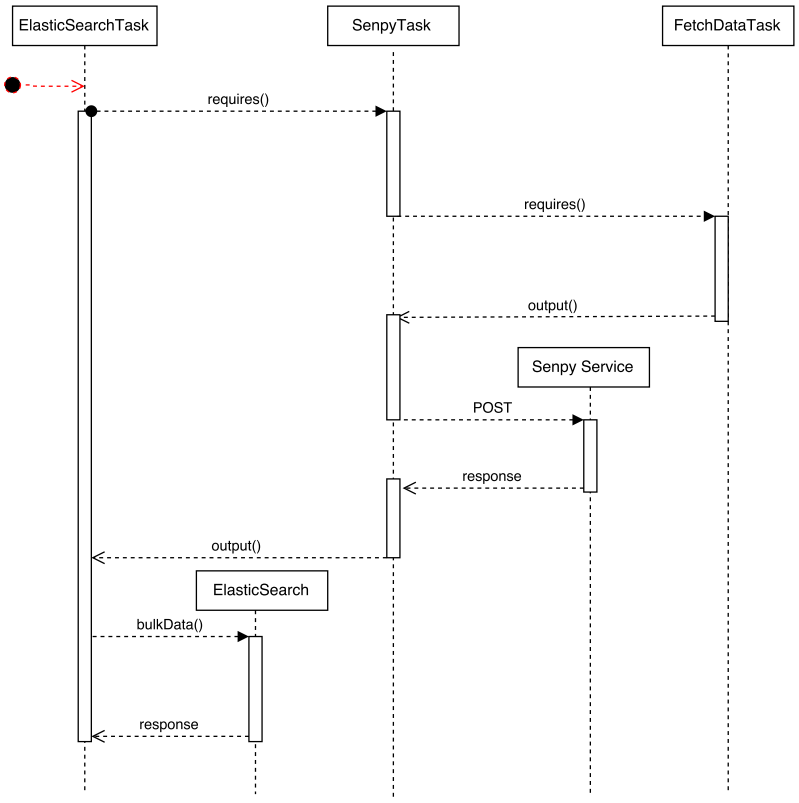


Figure 3.3: Create Workflow case use

CHAPTER 4

Architecture

4.1 Introduction

In this chapter, we cover the design phase of this project, as well as implementation details involving its architecture. Firstly, we present an overview of the project, divided into several modules. This is intended to offer the reader a general view of this project architecture. After that, we present each module separately and in much more depth.

The main purpose of this project is to obtain a framework to analyze data in which users can explore or filter data and make queries from a Sparql Engine. In addition, it can consult data from social web pages, in order to check if the info which appears in that website is true or is created by a bot. Furthermore, an admin can assign tasks, manage the workflow engine and create new tasks to be processed. Besides this, it manages dashboards. It means that it can change a dashboard, adding a new chart, new data, removing some charts or a dashboard, or simply creating a new one.

First of all, we need to present a web page with an automatic dashboard generated, with some data inside. Then a user could use the **Model-View** module to filter data through the way of aggregate searches or textual searches. This module is the main module of the project. However, we need a tool for filtering data, we call this tool **Filtering**, which will process those searches. But, it needs a module of persistence, a database which contains data stored in it, and **ElasticSearch** is that service. It can be implemented as server side technology or client side technology, depending the purpose of use.

With all the modules above, we have a platform that allows us to explore and analyze data presented in dashboards. Nevertheless, it is not able to search data making queries Sparql against a Fuseki Service or assign tasks, create new other tasks, or execute them. This is the reason to implement **Sparql Engine** and **Workflow Engine**. On the one hand, the first one extends the functionalities of Sparql Engine, to make possible create queries, execute them and check the response. On the other hand, Workflow engine allows us to extend the functionalities of Luigi. The admin could create new tasks, scheduled them and they will run automatically. Furthermore, that will be able to create auto analysis of live data, store them and make scrapers to retrieve other data indexed in a web page.

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

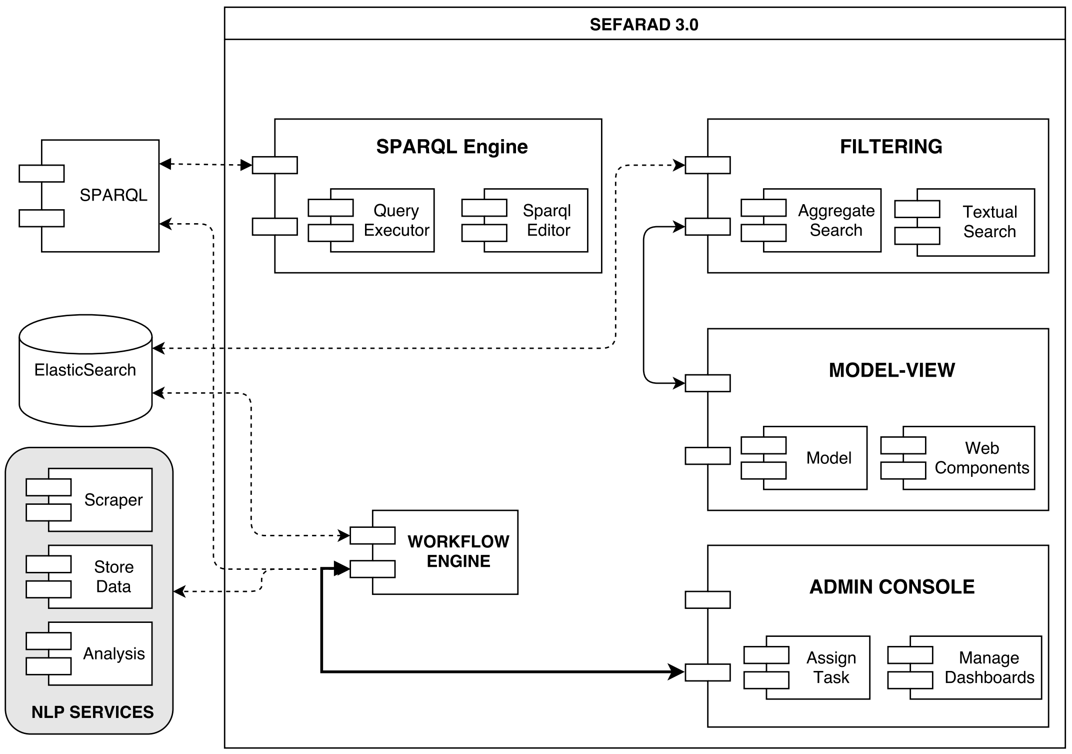


Figure 4.1: General Architecture

4.2 General overview

The core of this project is the **Model-View** module. As it can be seen in 4.1, Sefarad is composed by two ways: *Model-View* and *Filtering*, or *Sparql Engine*. This is in this way as a user. If you are an admin, there is another different way: *Admin Console* and *Workflow Engine*. All these modules are differentiated following a functional criteria. The interconnection of these modules into a major functionality is represented by the following flow diagram:

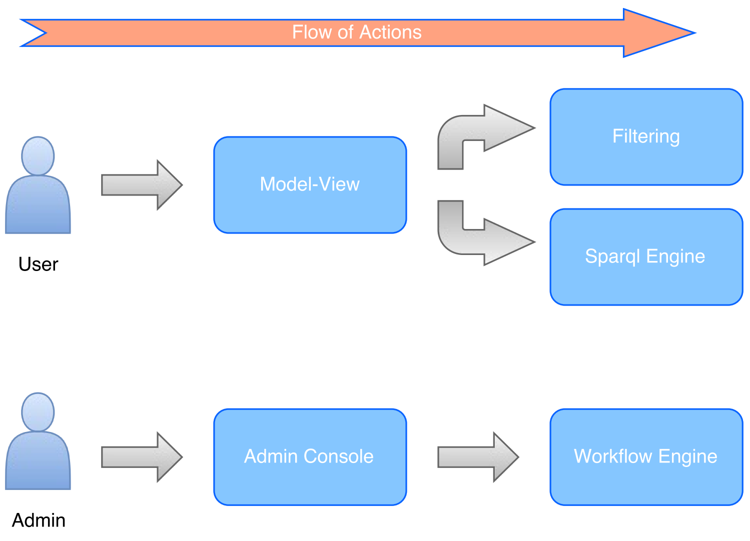


Figure 4.2: Modules Flow Diagram

In the figure 4.2 can be seen that the user interacts with the *Model-View* module. That is, the user choose between filter data among charts or create queries and execute them. The flow represented above is described as follows.

1. *Model-View.* The user interacts with dashboards or the web interface for analyzing data. This interface is primarily graphic, chart based. The main function of this module is to represent data which were processed and draw different charts to search interesting data. Furthermore, there is another part, a different tab where the user can create new queries and execute them, or default queries. Therefore, depending which way the user has chosen, Model-View will be redirected by two different paths.

2. *Filtering.* Once the user has filtered data in dashboards, Filtering receives it. The main function of this module is to make petitions REST to ElasticSearch indicating which filters will be activated and what kind of parameters will be sought by the user. Each petition is sent to ElasticSearch where data are stored. With this, the user could

only retrieve data it wants to.

3. *Sparql Engine.* In the same way, the user could access to YASGUI tab and explore data stored in Fuseki service, either executing default queries that Sefarad will provide or queries that the user has created. As a result, data have been retrieved will be displayed in a table with different formats (JSON, text, sortable table...).

It can be seen a different flow of actions from the admin. It interacts with the *Admin Console* module that has different tools to improve the framework, or simply adapt to the needs of new users. The flow represented in the figure 4.2 is described as follows.

1. *Admin Console.* The admin interacts with different tools that can be modified externally for improving the system, check functionalities of a database, represents or create new workflows. It is a web interface where these tools are represented by their logo, with a direct link to their path, their environment of execution.

2. *Workflow Engine.* Once the admin has chosen Luigi tool, it acceses to workflow engine where admin can change tasks will be executed, schedule new tasks, create new scrappers, in short, use Luigi or GSI Crawler with full functionality.

4.3 Model-View

Model View is the core of this project, one of the most important modules. This module provides an interface composed by two tabs. One of them is related to show many charts which are representing the data. During the whole project, the most effort has been made to elaborate this graphical interface, based on adapting charts with the capabilities of filtering, searching or showing data actions. These are intended to make easy and fast the process of creating or editing dashboards.

The other tab, is related to Sparql Engine, where a framework appears, enabling the execution of queries against an endpoint and it retrieves data with a semantic structure.

4.3.1 Graphic interface

As it is said, Model View interface is composed by two tabs. The first one, charts chosen by the admin or the developer will be shown. Different fields will be represented and it can be filtered either clicking in the value you want to or writing the value in a search field you want to. All widgets are made with Polymer framework as it will be explained later.

With this framework, an admin can simply add a new dashboard choosing the charts it wants and inserting the index and subindex of where data are stored in Elasticsearch and it will be deployed in that dashboard.

An example of this interface is presented in the figure 4.3.

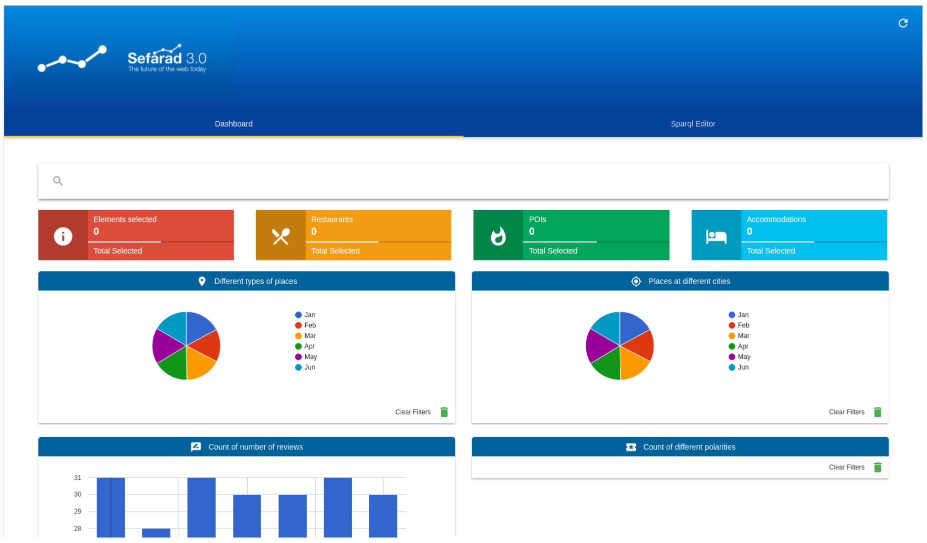


Figure 4.3: Dashboard web interface

All this logic is implemented using client-side: JavaScript, CSS, HTML5 and Polymer. Web components based on Polymer contain the code for filtering and retrieving data using the ElasticSearch library for JavaScript.

4.3.2 Web Components

Web components are a main feature of Sefarad 3.0, and in this section we will show how we exploit their capabilities.

All widgets in Sefarad 3.0 are web Components (used under the Polymer implementation) and, as they, they are ideally encapsulated and isolated from the rest of the code.

First, we have a special HTML to import all web Components.

Then, each widget is instantiated in the dashboard through its custom HTML tag. For example, if we have imported a wheel-chart widget and now we want to include it in our dashboard, the required code would be the one listed below (4.1).

```
<!-- wheel-search -->
<wheel-chart
  title="Wheel Chart Sentiment"
  icon="toll"
  query="{{query}}"
  index="ftten"
  subindex="entities"
  field=""
  fields='["user", "sentiment", "name", "text"]'
  filters="{{filters}}"
  param="{{param}}"
  host='{{endpoint}}'>
</wheel-chart>
<!-- /.wheel-search -->
```

Listing 4.1: Web Components Insertion

Reading carefully, it can be noticed that the entire widget is encapsulated inside the wheel-search tag, with its parameters set as tag parameters.

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.

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 a web component, including this bindings in the code listing 4.2.

```
<dom-module id="wheel-chart">
  <link rel="import" type="css" href="wheel-chart.css">
  <template>
    <div class="top-bar">
      <iron-icon icon="{{icon}}"></iron-icon>
      <span>{{title}}</span>
    </div>
    <div id="chart"></div>
    <div id="text">{{text}}</div>
  </template>
  <script>
    Polymer({
      is: 'wheel-chart',
      ...
    });
  </script>
</dom-module>
```

Listing 4.2: Web Component Structure

In 4.2 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.

4.4 Filtering

Each dashboard is enabled the option of filter data clicking the parameter you want to filter and the value you want to look for. When one of these ways is chosen a trigger will fire an action inside the web component depending on the way.

```
defaultQuery: function() {
  var client = new $.es.Client({
    hosts: this.host
  });
  client.search({
    index: this.index,
    type: this.subindex,
    body: {
      size: 10000,
      query: {
        match_all: {}
      }
    }
  }).then(function (resp) {
    ...
  }
},
_filtersChanged: function() {
...
},
_queryChanged: function() {
...
}
```

Listing 4.3: Filtering data function

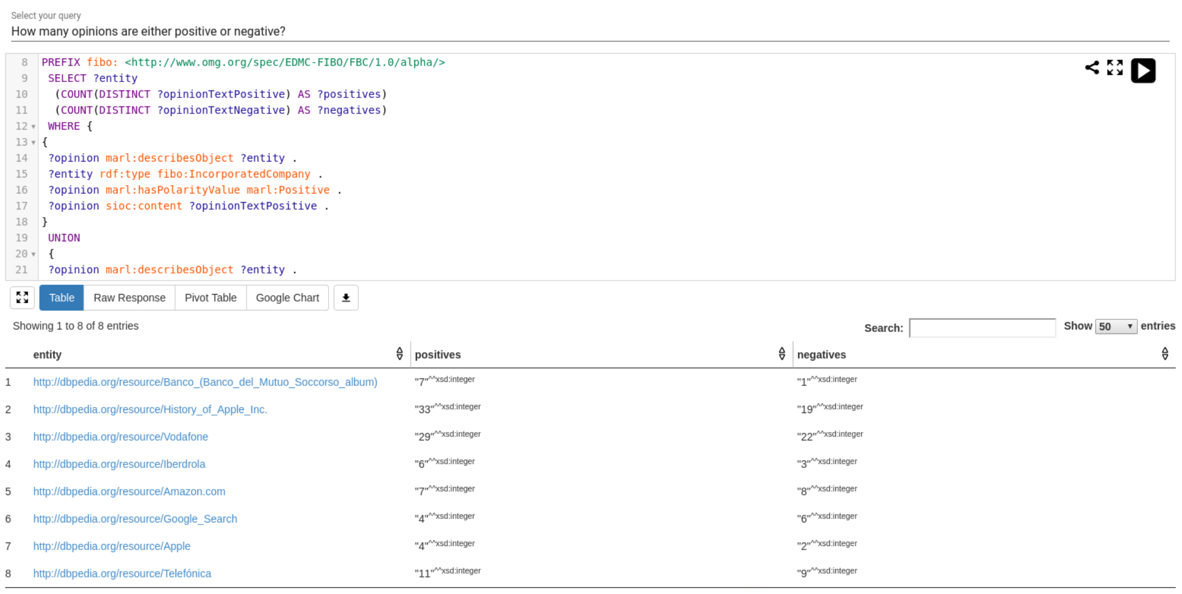
When the dashboard is loaded for the first time or all filters are removed, the *defaultQuery* function will be triggered, but, when a value is clicked, the function activated will be *_filtersChanged*.

Despite of this, at the top of the dashboard, a text editor is displayed and we can enter a natural language query to filter data. When it happens a different function is fired off. It is *_queryChanged*.

4.5 Sparql Engine

Each dashboard that we have created comes with default queries. These queries will be automatically executed when the dashboard is loaded and its results are rendered in the YASR widget.

But we have the choice to view and edit these default queries. In each dashboard we have the option to display the query editor, with a tab control at the top of the page as can be seen before. So, if we choose to see the query editor we will be prompted with this interface:



How many opinions are either positive or negative?

```
PREFIX fibo: <http://www.omg.org/spec/EDMC-FIBO/FBC/1.0/alpha/>
 SELECT ?entity
  (COUNT(DISTINCT ?opinionTextPositive) AS ?positives)
  (COUNT(DISTINCT ?opinionTextNegative) AS ?negatives)
 WHERE {
{
 ?opinion marl:describesObject ?entity .
 ?entity rdf:type fibo:IncorporatedCompany .
 ?opinion marl:hasPolarityValue marl:Positive .
 ?opinion sioc:content ?opinionTextPositive .
}
 UNION
 {
 ?opinion marl:describesObject ?entity .
```

Showing 1 to 8 of 8 entries

Search: Show 50 entries

	entity	positives	negatives
1	http://dbpedia.org/resource/Banco_(Banco_del_Mutuo_Soccorso_album)	"7"^^xsd:integer	"1"^^xsd:integer
2	http://dbpedia.org/resource/History_of_Apple_Inc.	"33"^^xsd:integer	"19"^^xsd:integer
3	http://dbpedia.org/resource/Vodafone	"29"^^xsd:integer	"22"^^xsd:integer
4	http://dbpedia.org/resource/Iberdrola	"6"^^xsd:integer	"3"^^xsd:integer
5	http://dbpedia.org/resource/Amazon.com	"7"^^xsd:integer	"8"^^xsd:integer
6	http://dbpedia.org/resource/Google_Search	"4"^^xsd:integer	"6"^^xsd:integer
7	http://dbpedia.org/resource/Apple	"4"^^xsd:integer	"2"^^xsd:integer
8	http://dbpedia.org/resource/Telefónica	"11"^^xsd:integer	"9"^^xsd:integer

Figure 4.4: Query Editor

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

4.6 Admin Console

The admin console panel allows for the admin manage the different tools of this framework. It can manage data with Kibana, a tool provided by the team who developed ElasticSearch and which contains different charts that can be modified as the admin wants to. Furthermore, the admin can manage tasks, creating new tasks, executing or scheduling tasks have just been created or remove the default tasks created by the developer. Besides this, Sefarad 3.0 allows admins to modify, create or remove dashboards and charts will be displayed on them. To conclude with this panel, the admin can manage senpy tool, choosing new algorithms or the language the text was written. When any of these tools are selected a new web page will be deployed.

In the figure 4.5 an image of all the tools is presented.

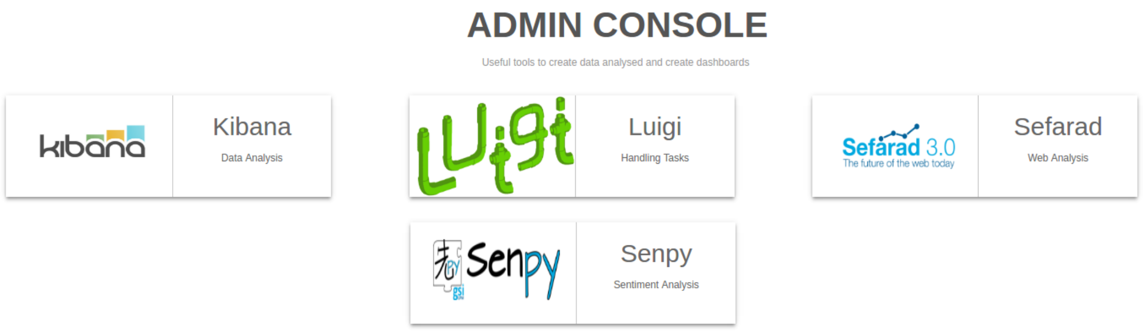


Figure 4.5: Admin Console Web

The main tool of this console is Luigi. It is the core of *Workflow Engine* and it helps the admin build complex pipelines of batch jobs. It handles dependency resolution, workflow management, visualization, handling failures, command line integration, and much more. It is more explained in the next section 4.7.

4.7 Workflow Engine

The *Workflow Engine* will be based on Luigi, a framework developed by Spotify. Once the admin selects Luigi tool, it can execute the workflow and schedules it. This module will be responsible for retrieving data from a database, scraper, or a simple file and send those data to a second task. It follows a structure as shown below 4.4.

```
class FetchDataTask(luigi.Task):

  def run(self):
    ...
    with self.output().open('w') as output:
      json.dump(j, output)
      output.write('\n')

  def output(self):
    return luigi.LocalTarget(path='/tmp/_docs-%s.json' % self.field)

class SenpyTask(luigi.Task):

  def requires(self):
    return FetchDataTask()

  def output(self):
    return luigi.LocalTarget(path='/tmp/analyzed-%s.jsonld' % self.file)

  def run(self):
    ...
    output.write(json.dumps(i))
    output.write('\n')
```

Listing 4.4: Filtering data function

The second task is for analyzing data in Senpy service. It sends data to the Senpy endpoint and get the response and store them in an output file. Finally, there are two different ways of exporting those data: ElasticSearch or Semantic File. On the one hand, if ElasticSearch will be selected, the data will be transformed to be indexed in it through LogStash, and the admin may select an index and a subindex to choose where data will be stored. It gives the path to the data to the admin. On the other hand, SemanticTask transform the response from Senpy tool from JSON-LD format to S3 format. Then this file transformed could be indexed in Fuseki service and be available from an endpoint to make queries against it.

This workflow follows the pattern shown in the figure 4.6:

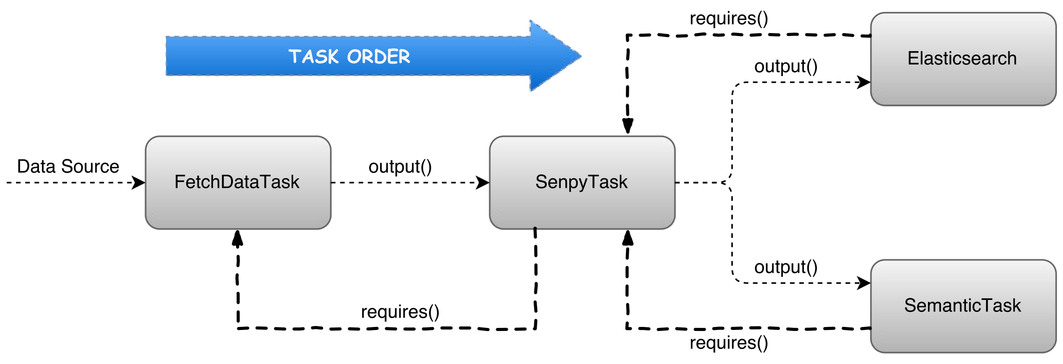


Figure 4.6: Admin Console Workflow

4.8 Widgets

Many of these Web Components created are widgets for data visualisation, we have used D3.js [1] to make them.

Each widget provides us different information based on tweet's data. Some of these widgets are described below.

4.8.1 Number Chart

We have created this widget to represent the number of restaurants, points of interest, or total number of data. This data is real time updated and is located in the main bar. The parameters accepted by this widget are:

- Index and Subindex: This is required to get the total number of places in each index.
- Query: This parameter is auto filled with search box to determine the number of data being analysed and studied.

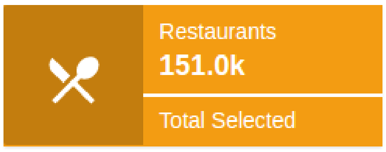


Figure 4.7: Number Chart Widget

4.8.2 Pie Chart

This widget represents the total number of a entity, how many data are retrieved from database. A pie is divided depending on the percentage total data of one entity. The parameters accepted by this widget are:

- Index and Subindex: This is required to get the total number of entities in each index.
- Query: This parameter is auto filled with search box to determine the number of entities being analysed and studied.

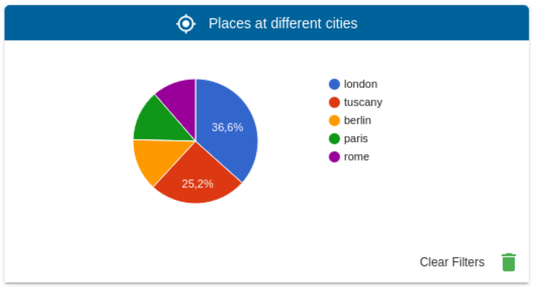
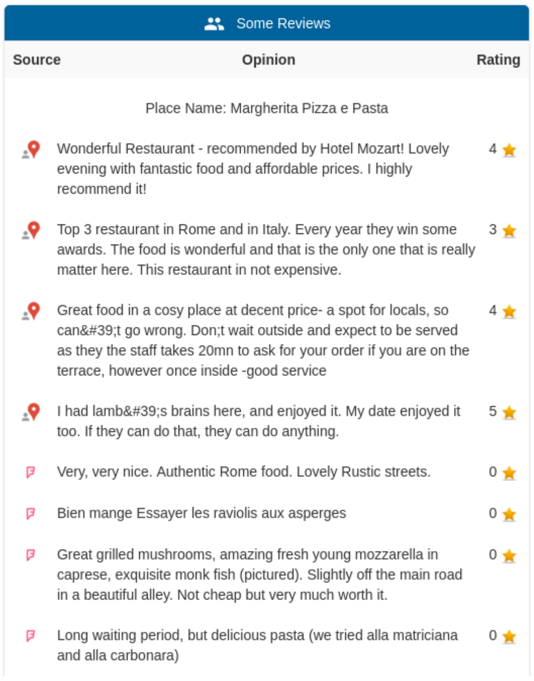


Figure 4.8: Pie Chart Widget

4.8.3 Number Chart

We have created this widget to represent reviews of restaurants, points of interest, or accommodations. This widget recognizes the social media it comes from and select the icon, and it shows the number of stars that was qualified. The parameters accepted by this widget are:

- Index and Subindex: This is required to get the total number of places in each index.
- Query: This parameter is auto filled with search box to determine the number of data being analysed and studied.



Some Reviews

Source	Opinion	Rating
	Place Name: Margherita Pizza e Pasta	
	Wonderful Restaurant - recommended by Hotel Mozart! Lovely evening with fantastic food and affordable prices. I highly recommend it!	4 ★
	Top 3 restaurant in Rome and in Italy. Every year they win some awards. The food is wonderful and that is the only one that is really matter here. This restaurant in not expensive.	3 ★
	Great food in a cosy place at decent price- a spot for locals, so can&#39;t go wrong. Don;t wait outside and expect to be served as they the staff takes 20mn to ask for your order if you are on the terrace, however once inside -good service	4 ★
	I had lamb&#39;s brains here, and enjoyed it. My date enjoyed it too. If they can do that, they can do anything.	5 ★
	Very, very nice. Authentic Rome food. Lovely Rustic streets.	0 ★
	Bien mange Essayer les raviolis aux asperges	0 ★
	Great grilled mushrooms, amazing fresh young mozzarella in caprese, exquisite monk fish (pictured). Slightly off the main road in a beautiful alley. Not cheap but very much worth it.	0 ★
	Long waiting period, but delicious pasta (we tried alla matriciana and alla carbonara)	0 ★

Figure 4.9: Review Chart Widget

4.8.4 Map Chart

Many of these data we have retrieved from Tourpedia and analyze can be located thanks to location of the place or user's location.

We have used this location to place data in a map of Europe. This map has been designed with D3.js. The parameters accepted are:

- Index and Subindex: Indicates in which Elasticsearch's database are you querying.
- Query: This parameter is auto filled with search box.

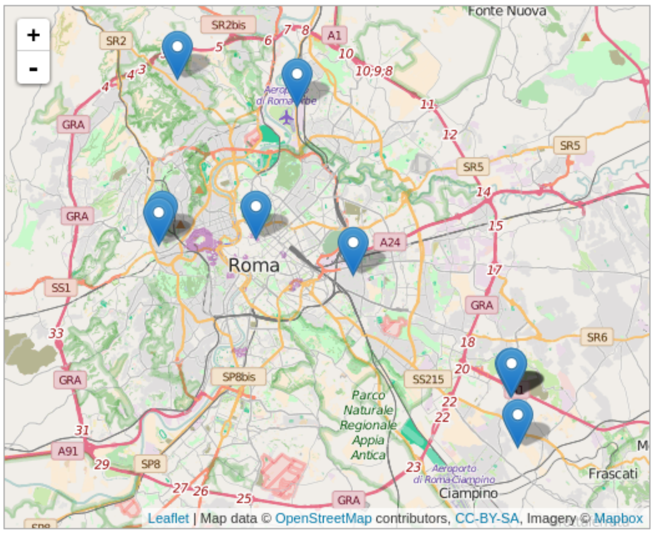

Roma
Fonte Nuova
Frascati
Ciampino
Parco Naturale Regionale Appia Antica
Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox

Figure 4.10: Map Chart Widget

4.8.5 Stock and Values Chart

On the one hand, sentiment value chart is a line chart with the evolution of sentiments along of time, representing the number of positive or negative opinions from users. It can approximate the future value of its stock value. On the other hand, stock value chart represents the real evolution of the price of stock value of a certain company along of time. The parameters accepted are:

- Index and Subindex: Indicates in which Elasticsearch's database are you querying.
- Query: This parameter is auto filled with search box.
- Show: Selects which value you want to show.

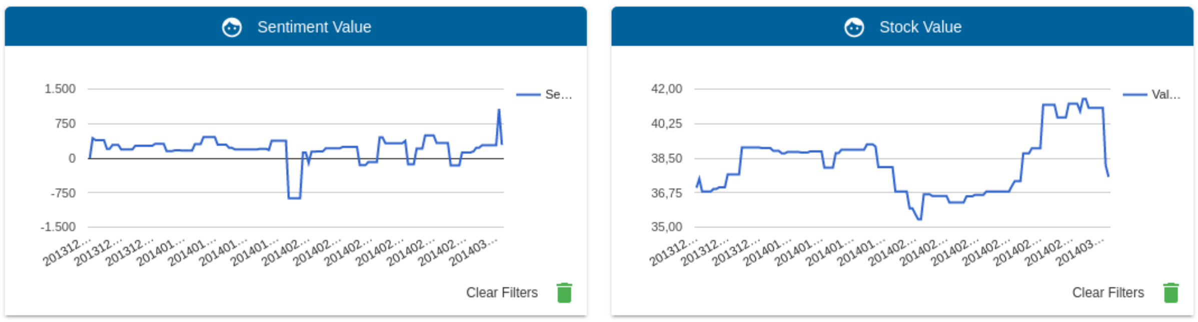


Figure 4.11: Stock and Values Chart Widget

4.8.6 Chernoff Faces Chart

Chernoff faces [6], display multivariate data in the shape of a human face. This chart draw those faces, separating by individual parts, such as eyes, ears, mouth and nose that represent values of the variables by their shape, size, placement and orientation. The idea behind using faces is that humans easily recognize faces and notice small changes without difficulty. This faces has been designed with D3.js. The parameters accepted are:

- Index and Subindex: Indicates in which Elasticsearch's database are you querying.
- Query: This parameter is auto filled with search box.

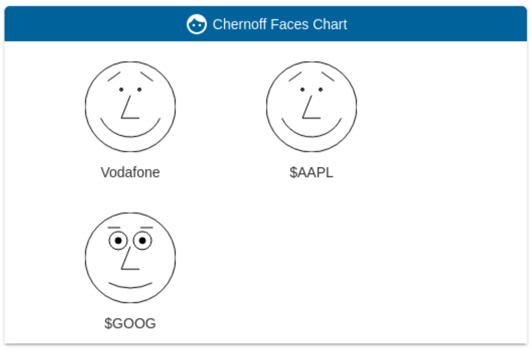
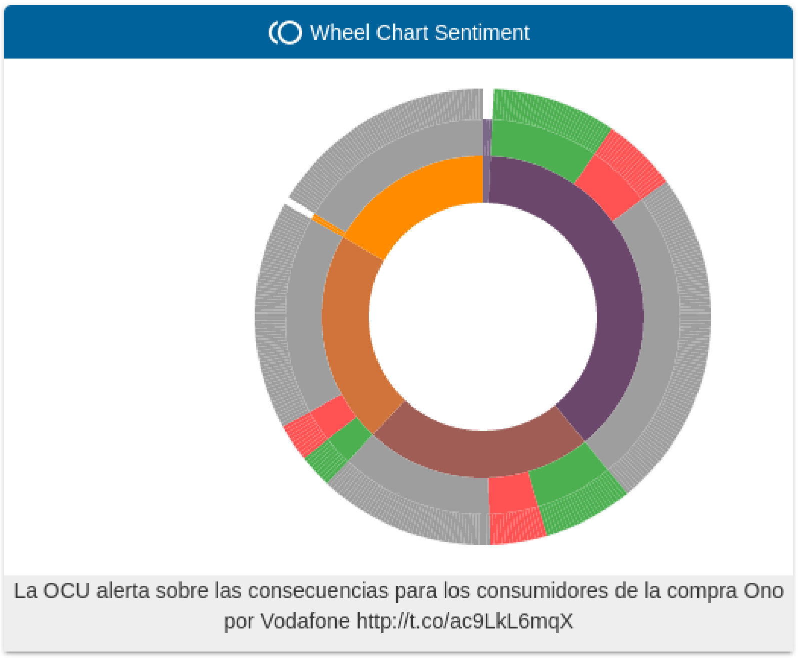


Figure 4.12: Chernoff Faces Chart Widget

4.8.7 Wheel Chart

This widget represent a wheel divided by companies, and each one are divided by sentiments. Inside those sentiments, are drawn different opinions from users and filled depending on the sentiment: green for positive opinions, red for negative opinions and gray for neutral opinions. The parameters accepted are:

- Index and Subindex: Indicates in which Elasticsearch's database are you querying.
- Query: This parameter is auto filled with search box.



La OCU alerta sobre las consecuencias para los consumidores de la compra Ono por Vodafone http://t.co/ac9LkL6mqX

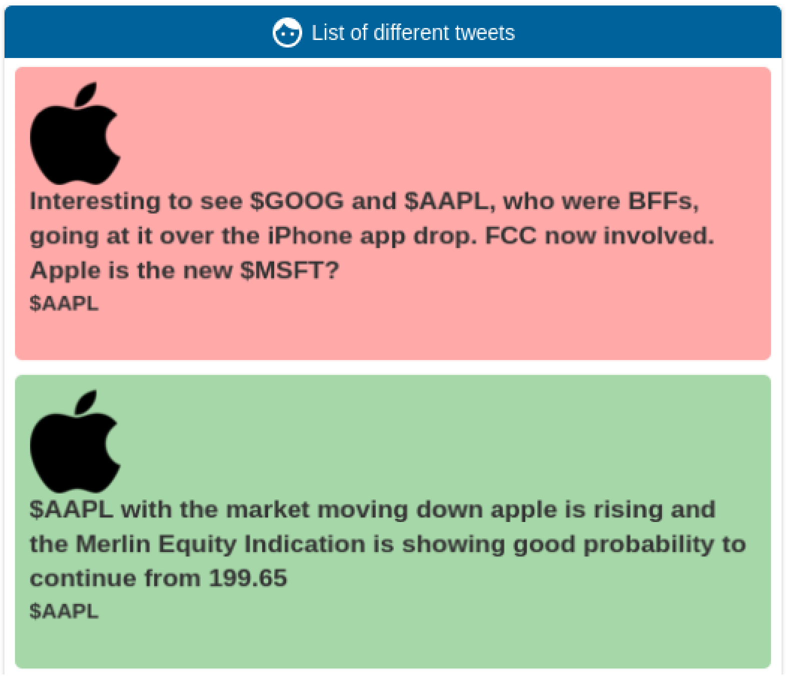
Figure 4.13: Wheel Chart Widget

4.8.8 Tweet Chart

This widget is used for showing the latest tweets available. The parameters accepted are:

- Index and Subindex: Indicate in which Elasticsearch's database are you querying.
- Query: This parameter is auto filled with search box.

Data received is presented on a list, tweet's background is coloured according to each tweet sentiment: green colour portrays a positive tweet, red colour represents a negative tweet and gray colour means a neutral tweet.


List of different tweets

Interesting to see $GOOG and $AAPL, who were BFFs, going at it over the iPhone app drop. FCC now involved. Apple is the new $MSFT?
$AAPL

$AAPL with the market moving down apple is rising and the Merlin Equity Indication is showing good probability to continue from 199.65
$AAPL

Figure 4.14: Tweet Chart Widget

CHAPTER 5

Case study

In this chapter, we present the two use cases we have applied to Sefarad 3.0.

The first one is the redesign of the Sefarad 2.0 demonstration for the Tourpedia use case. This case will be centred in the development of tools for rendering thousands rows of data. We will analyse the facets lackness of this dataset, giving ideas of how to solve it, and how to represent reviews of places which are filtered.

The second case shares a dataset with another project of GSI Group. That project is Financial Twitter Tracker. Its main objective is the enrichment of financial content with information extracted from social media Twitter, as well as detection of demand for new financial content in certain topics. We will try how to implement it with our system, new charts, more fluid and more enhanced than the originl FTT project.

5.1 Tourpedia Dashboard

TourPedia is the result of an European project. It is a demo of OpenNER[1] (Open Polarity Enhanced Name Entity Recognition). It contains information about accommodations, restaurants, points of interest and attractions of different places in Europe. At the moment

[1]http://www.opener-project.eu/

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 analyze how improve the demostration of the previous version. We detected that the dataset was reduced to only two cities to get more handable data, but it is unsupportable in the future when TourPedia will be extended to all the world. In this way, we also implement all type of places to be more consistent with the data and dashboard.

5.1.1 Structure and pre-process

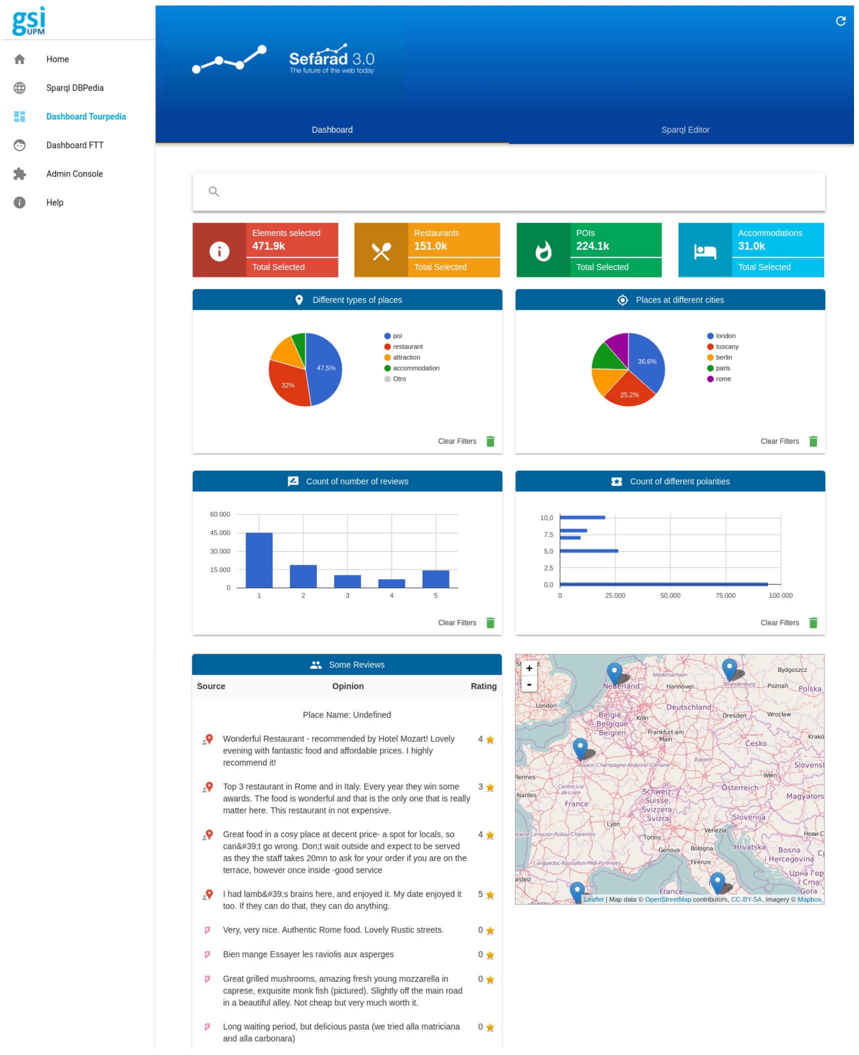
The data is a collection of around 500.000 features with a rich set of facets.

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.

5.1.2 Analysis Design

We have analyzed the information about this rich dataset in the following way:


gsi UPM
Home
Sparql DBPedia
Dashboard Tourpedia
Dashboard FTT
Admin Console
Help
Sefarad 3.0
The future of the web today
Dashboard
Sparql Editor
Elements selected 471.9k Total Selected
Restaurants 151.0k Total Selected
POIs 224.1k Total Selected
Accommodations 31.0k Total Selected
Different types of places
poi
restaurant
attraction
accommodation
Otro
47.5%
32%
Places at different cities
london
tuscany
berlin
paris
rome
36.6%
25.2%
Clear Filters
Count of number of reviews
Count of different polarities
Some Reviews

Source	Opinion	Rating
	Place Name: Undefined	
	Wonderful Restaurant - recommended by Hotel Mozart! Lovely evening with fantastic food and affordable prices. I highly recommend it!	4 ★
	Top 3 restaurant in Rome and in Italy. Every year they win some awards. The food is wonderful and that is the only one that is really matter here. This restaurant in not expensive.	3 ★
	Great food in a cosy place at decent price- a spot for locals, so can&#39;t go wrong. Don;t wait outside and expect to be served as they the staff takes 20mn to ask for your order if you are on the terrace, however once inside -good service	4 ★
	I had lamb&#39;s brains here, and enjoyed it. My date enjoyed it too. If they can do that, they can do anything.	5 ★
	Very, very nice. Authentic Rome food. Lovely Rustic streets.	0 ★
	Bien mange Essayer les raviolis aux asperges	0 ★
	Great grilled mushrooms, amazing fresh young mozzarella in caprese, exquisite monk fish (pictured). Slightly off the main road in a beautiful alley. Not cheap but very much worth it.	0 ★
	Long waiting period, but delicious pasta (we tried alla matriciana and alla carbonara)	0 ★

Figure 5.1: Dashboard of Tourpedia

In the first row of widgets we have four number charts. These are: total elements selected, Restaurants (value type), POIs (value type) and Accommodation (value type) which are selected.

In the second row we have two pie chart widgets, one of them is filtering how many types of places are different and the other is filtering how many quantity of them are selected at that moment.

In the third row we have two bar chart widgets, they are representing the count of different polarities and different reviews exists.

At last, we have implemented a custom chart, that shows a list of reviews of the ten most important places of data filtered. Apart from that, a map is implemented showing those places in it.

5.1.3 Conclusions

This dataset is full of information, with a rich set of facets for each feature and has a great number of features.

Due to these facts, we accept it as a final demonstration dataset. A possible future work over this dataset would be storing the complete data, including reviews of each datum, on the server side and serve only the relevant data to the Sefarad 3.0 application, so the possibility of being extended to the whole world could be ready when it will be released.

5.2 FTT Dashboard

Financial Twitter Tracker is an R&D project of GSI Group that needs to be remastered to offer full funtionalities and properties it must to have. It contains information about people talking about brands in social media like Twitter, Facebook, and more.

This project contains one dataset separated in several different files depending on brands. Those files are composed by text without analyzed, person who wrote that text, the id of the account, where that text was written, and some different results from different algorithms.

These data are stored in ElasticSearch or can be loaded one by one from Json files. GSI Group are enhancing this to transform them to RDF file. Then, it could be queried from Query Editor tab.

At first we analyze how improve the demostration of the previous version. We detected that some functionalities are only enabled when the quantity of data was reduced, furthermore, this affects to load a reduced dataset and the results could not to be right. In this way, implement ElasticSearch and Web Components technologies will be enough to improve this project.

5.2.1 Structure and pre-process

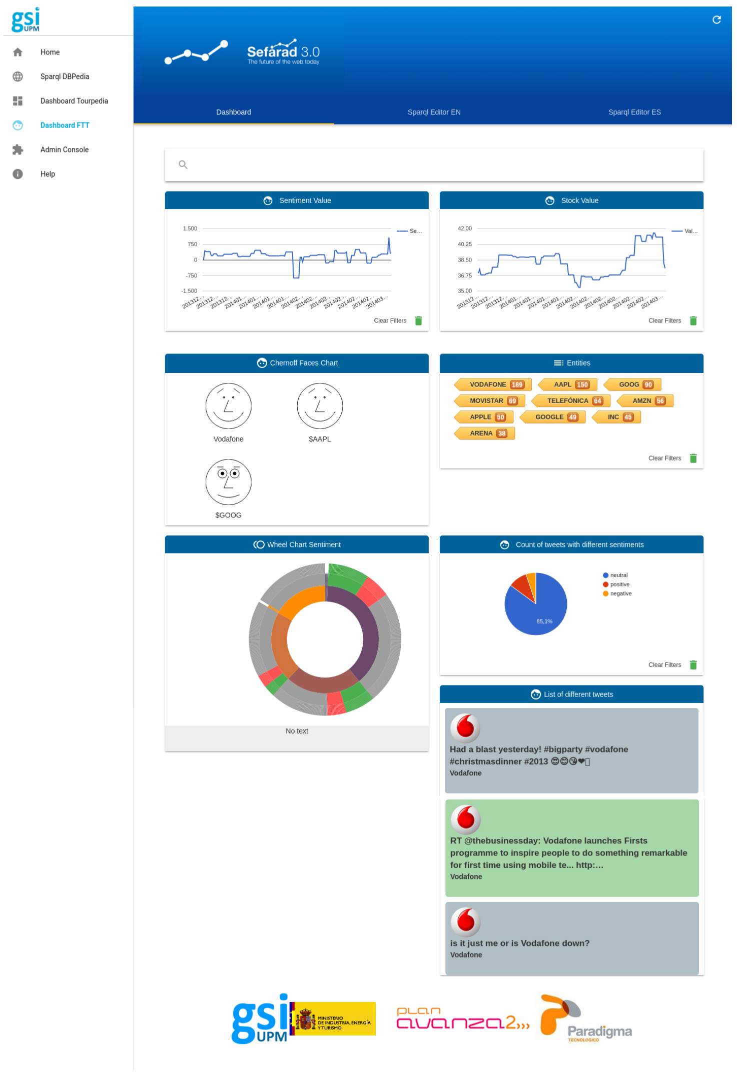
The data is a collection of 7 important companies with a rich set of facets.

In these data, we can found the following information:

- **Id**: The unique id of each comment.
- **Date**: Date comment was written.
- **Paradigma**: Results of sentiment analysis using paradigma algorithm. Only spanish dataset.
- **Paradigma**: Results of sentiment analysis using paradigma algorithm. Only english dataset.
- **Price**: Stock value of that moment.
- **Return**: Stock value after the analysis.
- **Emotions**: Resulst of emotion analysis.

5.2.2 Analysis Design

We have analyzed the information about this rich dataset the following way:



Had a blast yesterday! #bigparty #vodafone #christmasdinner #2013 😊😊😘❤
Vodafone

RT @thebusinessday: Vodafone launches Firsts programme to inspire people to do something remarkable for first time using mobile te... http:...
Vodafone

is it just me or is Vodafone down?
Vodafone

Figure 5.2: Dashboard of FTT

In the first row of widgets we have two line charts. These are: stock values along of time without analysis, and approximate stock values following the results of analysis.

In the second row we have one Chernoff faces chart and one entities chart widgets. The first one is a custom chart that shows the emotions of one entity, or an approximation among them, in a human face. The other one represents a list of ten most important entities of the dataset.

At last, we have a wheel chart and a pie chart widgets, they are representing the quantity of sentiment in two different ways, dividing by companies and sentiments, also the text of that sentiment could be seen below of this chart, and how many comments exist of that sentiment.

5.2.3 Conclusions

This dataset is full of information, with a rich set of facets for each algorithm using for analyzing.

Due to these facts, we accept it as a final demonstration dataset. A possible future work over this test would be join into only one chart the stock value charts of the first row. Chernoff faces works properly, however, it is not the real Chernoff faces because of problems implementing the library. It could be another possible work to enhance this test.

CHAPTER 6

Conclusions and future work

In this chapter we will describe the conclusions extracted from this project, and the thoughts about future work.

6.1 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 Financial Twitter Tracker project contributing to this project. 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, luigi workflow and d3.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 learnt from past experiences developing for Sefarad 3.0, 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 Polymer and Luigi frameworks, fixing bugs and creating new functionalities. We have found the need to test the tool in order to find failures and possible improvements. Some of our modules and developments are the result of experimentation and detection of new needs.

6.2 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.

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 3. We have gone through three different use cases, analysing each one and evaluating their pros and cons until we have reach a successful stage.

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 4.

Document the final application to ease future developments or use cases. We have committed the Chapter 5 to the detailed explanation and documentation of some use cases of Sefarad 3.0 for future Sefarad developers.

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 3.0 and an auto indexer of data to ElasticSearch.
- Add widgets on the fly, developing a new set of graphical tools for the selection and parameters settings.
- Apply Sefarad 3.0 to new projects where data graphical analysis is needed, developing new custom widgets in order to face new problems.
- Create custom dashboard on the fly, selecting parameters to filter and saving the configuration in ElasticSearch.
- Create new Luigi workflows for auto update data to Fuseki apart from create an N3 file, and return the endpoint to the admin.
- Retrieve and analyze data in realtime with logstash, storing it in ElasticSearch. It can be possible thanks to Luigi.
- Create this project into one container using Docker.

Sefarad 3.0 is still young. As developers, we can find a lot of new functionality examples in older web and tools as Influence Tracker [11], Kibana, 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.

Bibliography

[1] Mike Bostock. D3.js is a JavaScript library for manipulating documents based on data. Documents. En Línea. Disponible en: `https://d3js.org/`.

[2] José Emilio Carmona. Development of a Social Media Crawler for Sentiment Analysis., 2015.

[3] Clinton Gormley and Zachary Tong. Elasticsearch - Search and Analyze Data in Real Time., 2016. En Línea. Disponible en: `https://www.elastic.co/products/elasticsearch`.

[4] Clinton Gormley and Zachary Tong. Kibana - Explore and Visualize Your Data., 2016. En Línea. Disponible en: `https://www.elastic.co/products/kibana`.

[5] Clinton Gormley and Zachary Tong. Logstash - Collect, Enrich and Transport Data., 2016. En Línea. Disponible en: `https://www.elastic.co/products/logstash`.

[6] Lars Kotthof. Chernoff faces, display multivariate data in the shape of a human face. En Línea. Disponible en: `http://bl.ocks.org/larskotthoff/2011590`.

[7] Luigi. Workflow mgmgt + task scheduling + dependency resolution., 2011. En Línea. Disponible en: `https://pypi.python.org/pypi/luigi`.

[8] Paradigma. Financial Twitter Tracker, sistema de análisis y evaluacion financiera, 2011. En Línea. Disponible en: `https://www.paradigmadigital.com/portfolio/financial-twitter-tracker/`.

[9] Ignacio Corcuera Platas. Development of an Emotion Analysis System in a University Community., 2015.

[10] Jorge Pérez and Marcelo Arenas. Querying semantic web data with SPARQL: State of the art and research perspectives. En Línea. Disponible en: `http://web.ing.puc.cl/~marenas/talks/pods11.pdf`.

[11] Gerasimos Razis and Dr Ioannis Anagnostopoulos. Discover the Influence Metric and various other characteristics of a Twitter account. En Línea. Disponible en: `http://www.influencetracker.com/`.

[12] Tourpedia. TourPedia is the Wikipedia of Tourism., 2013. En Línea. Disponible en: `http://tour-pedia.org/about/`.

[13] Webcomponents. WebComponents.org a place to discuss and evolve web component best-practices. En Linea. Disponible en: `http://webcomponents.org/`.