UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DEVELOPMENT OF A MONITORING DASHBOARD FOR SENTIMENT AND EMOTION IN GEOLOCATED SOCIAL MEDIA

JORGE GARCÍA CASTAÑO

2017

TRABAJO DE FIN DE GRADO

Título:	Desarrollo de un panel de monitorización de sentimientos y emociones en redes sociales geolocalizadas
Título (inglés):	Development of a monitoring dashboard for sentiment and emotion in geolocated social media
Autor:	Jorge García Castaño
Tutor:	Juan Fernando Sánchez Rada
Departamento:	Grupo de Sistemas Inteligentes

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

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 MONITORING DASHBOARD FOR SENTIMENT AND EMOTION IN GEOLOCATED SOCIAL MEDIA

Jorge García Castaño

Julio de 2017

Resumen

Esta memoria es el resultado de un proyecto cuyo objetivo es diseñar y desarrollar un panel con una interfaz gráfica muestre información sobre el análisis de sentimiento de los tweets publicados en una zona geografica. El elemento principal de este panel es un mapa de calor que mostrará en forma de gradiente la variación de los valores obtenidos en el análisis de sentimiento y, por otro lado, la clasificación emocional de los tweets.

Ademas del mapa de calor, el dashboard debe estar preparado para incluir mas elemntos que expandan sus funcionalidades. Este proyecto ofrece dos: uno que muestra el listado de todos los tweets que se estan mostrando y otro que permite hacer un filtrado por fechas de los mismos.

Todos estos componentes requieren la obtencion y tratamiento de grandes cantidades de datos, lo cual también se encuentra recogido en este documento. Desde la obtencion de los tweets hasta la provisión de datos al panel, se ha definido un *pipeline* de procesos que realiza varias transformaciones sobre ellos, con intención de simplificar la carga de trabajo que debe hacer la aplicación web y su velocidad de interactuación con el usuario. Cabe recalcar que todo el desarrollo se ha hecho siguiendo la arquitectura definida por el proyecto Sefarad, mantenido por el Grupo de Sistemas Inteligentes.

Por último, recogemos las conclusiones extraídas del proyecto, las tecnologías que hemos aprendido durante el desarrollo del mismo, los problemas encontrados y las posibles líneas de trabajo futuro en relación con la continuación de este proyecto.

Palabras clave: Sefarad, dashboard, Polymer, WebComponents, Luigi, ElasticSearch, Javascript, Python, Leaflet, Twitter, big data, Senpy

Abstract

This thesis comes from a project which goal is to design and develop a dashboard with a graphic interface that shows information about the analysis of tweets posted in a geographic zone. The dashboard's main element is a heatmap that shows the value variation of certain sentiment parameters with a color gradient. Furthermore, this map will also clasify emotionally all analyzed data.

Besides the map, the dashboard has to be ready to host more elements that may increase its functionality. This project offers two: a list of all tweets being displayed, and a component which allows filtering the dataset by date.

All this elements require getting and preparing a large amount of data, which is also covered in this document. Since tweet fetching until dashboard data provision, a batch processing pipeline that makes several transformations to data has been defined. This is because it is interesting to simplify the web's workload and imporve user interaction speed. It is worth mentioning that this development has been made following the architecture defined by Sefarad project, mainained by *Grupo de Sistemas Inteligentes*.

Finally, there are more topics included in this document, such as final conclusions, learnt technologies, problems found during it, and possible future lines of work.

Keywords: Sefarad, dashboard, Polymer, WebComponents, Luigi, ElasticSearch, Javascript, Python, Leaflet, Twitter, big data, Senpy

Agradecimientos

A mis padres, por siempre haberme ayudado a cumplir a mis metas.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Context

As Internet connection capabilities increase in the world, so does user interaction with social networks. Hence, data analysis has become a main ingredient to big data commercial exploitation. Nowadays there are petabytes of raw information in all worldwide servers, which are not suitable for actual human analysis. Some data-based decision making applications are entirely made by AI, but some cases need human operational analysis. Therefore, if a human-friendly comprehension of certain data is wanted, there will be a necessity of data synthesis and clear visualization. The solution of this problem is a dashboard.

A dashboard is a graphic representation of key points which are important for certain decision making analysis, especially those related to enterprise. Some aspects are very important for the creation of a useful dashboard, such as data accuracy, simple visualization, data comparison, and customization capacity. Thus, as it would be expected, it is difficult to find a design that accomplish all this points.

Of course, a single dashboard is useless without data to back it up. Data analysis is a particular matter that can be different for each use case, as it solves specific data related problems. A very relevant kind of data analysis that is essential in this thesis is sentiment and emotional analysis. Sentiment analysis aims to provide synthesized sentiment and emotional data in order to understand social behavior. The analysis tends to be focused on specific domains or environments: cities, catastrophes, crises, celebrations... The analysis involves several processing methods, such as natural language processing, text analysis or computational linguistics.

Some developments with similar goals are *Geography of Hate*[14] or *Hedonometer*[9] projects.

1.2 Project goals

This project aims to provide a customizable dashboard to visualize emotion and sentiment in geo-located social network data. All data will be fetched from Twitter and analyzed through Senpy, a framework for sentiment and emotion analysis services. However, the dashboard would not be limited to the scope of this project itself, since the it is friendly with the introduction of new widgets developed in the future using simple interfaces.

The main goals for this project are the creation of:

- A dashboard that launches custom data queries to ElsaticSearch and distributes data to widgets. This component must follow the Sefarad interface for data sharing among widgets.
- A map widget that fetch analyzed data and shows a sentiment bsed heatmap and an emotion based Emoji visualization.
- A widget that shows all tweets that are being analyzed within the dashboard.
- A widget that allows to filter tweets by date in a graphic mode.
- A processing pipeline which can perform all the necessary operation in ElasticSearch to serve the data to the dashboard. This also include data fetching from Twitter and tweet annotation following the analysis provided by Senpy engine.
- A self-contained (dockerized) demo that can be run easily in any local environment.

1.3 Structure of this document

The document is structured as follows:

Chapter 1 provides an introduction to the context in which this project is developed. Besides, it describes the main objectives to achieve once concluded. ***Chapter 2*** offers a description of the main standards and technologies on which this project rely. ***Chapter 3*** describes a brief requirement analysis of the system of this project. ***Chapter 4*** describes the complete architecture of the system, decomposing it into several modules that will interact between them. ***Chapter 5*** offers an overview of use cases and ***Chapter 6*** sums up the conclusions extracted from this project, and we offer a brief view about the lines of future work.

CHAPTER 2

Enabling Technologies

2.1 Analysis and annotation

2.1.1 Emotional Analysis

Senpy[12] is a sentiment and emotion analysis server in Python that allows to get sentiment and emotion analysis with REST API requests. There are several plugins available to use with Senpy, e.g. different analysis methodologies which will affect to our analyzing tweets. Figure 2.1 summarizes how analysis process works. Once the JSON response is received, it is possible to process this data in every system or language.

In this work we will use the emotion-anew plugin, which applies parameters that measure the *Affective Norms for English Words*[15] (ANEW) metrics to its analysis. Also, it provides a distinctive emotion tag between: joy, sad, negative-fear, disgust and anger. This tag is given thanks to the influence of three certain parameters:

- **Valence**: pleasantness of a stimulus
- **Arousal**: intensity of emotion provoked by a stimulus
- **Dominance**: degree of control exerted by a stimulus

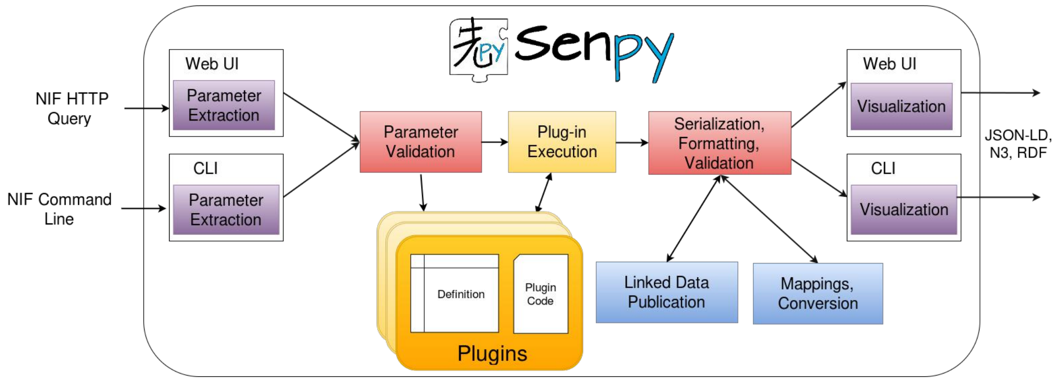
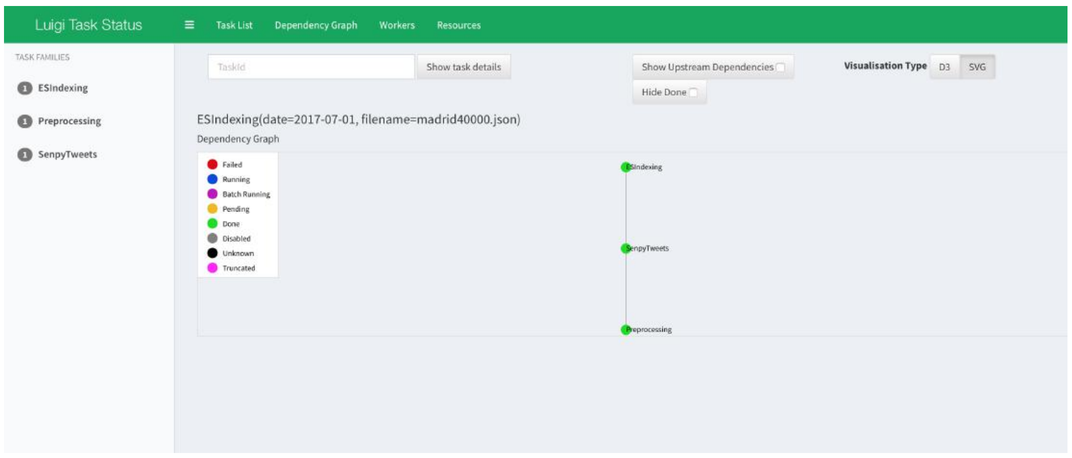


Figure 2.1: Senpy Architechture

2.2 Luigi

Luigi[11] is a Python library that allows to run batch pipelines with several features. It informs about execution failures, missing dependencies, failed steps... during the task execution. Luigi is useful for this project since it lets encapsulate sequences of function calls such as pushing into a database, fetching information from the Internet, or transforming certain database registry.



ESIndexing(date=2017-07-01, filename=madrid40000.json)

Dependency Graph

Figure 2.2: Web UI of the Luigi Central Scheduler

Furthermore, Luigi has other great features when running a batch pipeline, such as skipping already done tasks if they are dependencies for the current one. This accelerates the process if before some previous steps for certain task were done.

Besides, Luigi has a scheduler which can be used intensively in development phase. With this, it is possible to make sure two instances of the same task are not running simultaneously, and get a visualization of everything which is going on.

2.3 Storage

As all project data ingestion is locally or remote storage oriented. The chosen database system has been ElasticSearch.

2.3.1 ElasticSearch

ElasticSearch[1] is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases. This product offered by Elastic is based in Lucene [2] and is characterized by the use of *documents* instead of conventional tables. The deletion of the table concept means the inclusion of JSON. Every register in the database is done saving a JSON object in which each key would be the equivalent to a table column, and each value, to the own cell.

When a query is made, it directly flows through all saved documents with an *inverse indexing* technique, which traduces in fractions of seconds for million of documents.

Thanks to the inclusion of a so spread standard like JSON, there is a wide availability of database clients for every mainstream language. Anyway, curl it is always an option as ElasticSearch responds to a REST API [**?**].

Some of the features that makes ElasticSeach more interesting for this project over other database solutions are:

- **Search function**. The search API allows to pass an easily formatted JSON and get back search hits that match the query, which can either be provided using a simple query string as a parameter, or using a request body. This calls can be applied across multiple types within an index, and across multiple indices.
- **Scalability**. This project only uses one ElasticSearch server, but in a future it could use a cluster of them. One server can hold one or more parts of one or more indexes. Every such index, or part of it, is called a shard, and Elasticsearch shards can be moved around a cluster very easily. This means that, all index shards can be located in the same machine or not, fact that gives flexibility for distributed data processing.
- **Active community and development**. Such a popular technology results on a big community of users. It is very simple to find custom clients, implementation in other

[1] `http://elastic.co/`

[2] A Java text search engine library maintained by Apache Foundation (https://lucene.apache.org/core/)

technologies, loads of information... Elastic has a quality documentation and keeps on the development of its products in a constant basis.

2.3.2 JSON lines

This project has been tested and developed in a local machine that launches an ElasticSearch dockerized service, so a persistent local storage was necessary to stock all our testing data. The solution provided is a file containing lots of JSONs separated by a */n* character. This help us to index JSONs to ElasticSearch indexes directly pointing our indexing pipeline to this file.

2.4 Visualization

The visualization technologies used are the front-end core of all the user experience. Several technologies has been selected to get the maximum value of open source software and to ensure a modular development.

2.4.1 Polymer and WebComponents

The main purpose of the WebComponents[8] standard is the creation of HTML independent and reusable modules that can be imported very easily to any web project (with a Javascript engine), and can work independent of the app which is integrated on. A component like this usually receives some input parameters, expose other ones based on that inputs, and show a certain user interface to interact with. As it can be expected, this idea leads to a very big community that can provide lots of fully auto-functional components.

Polymer[7] is a JavaScript library developed by Google that let build progressive modern apps taking advantage of the WebComponents technology. Polymer adds some syntactic sugar that make a component easier to develop and also includes some interesting features:

- **Data binding and events**. Each custom component can share certain resources and events with other components and the main app. Thus, it helps to make the web app completely responsive to user interaction and data ingestion.
- **Shadow DOM**. Every component is composed of a Shadow DOM, that implies a fully functional HTML space, in which declare conventional HTML features and Polymer

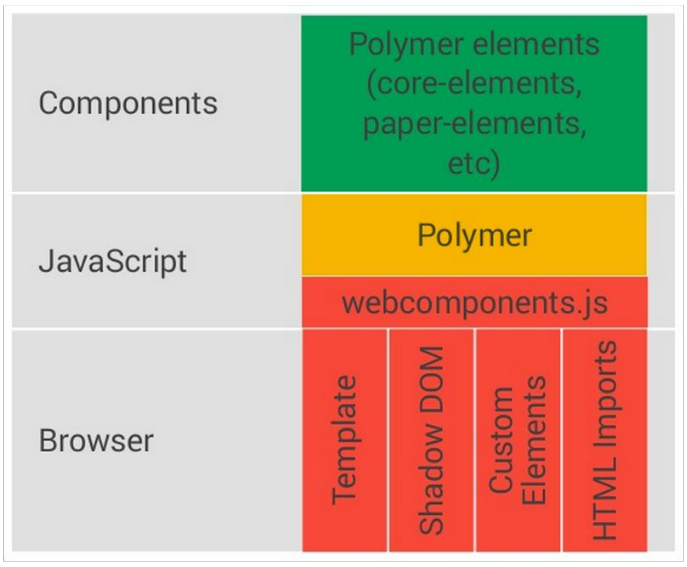


Figure 2.3: Polymer Web Stack

directives such as iterable or conditional tags. Also, more Polymer elements can be nested inside a shadow DOM.

- **Support to all browsers**. Standard WebComponents saves some functionalities to Firefox and fully crosses out Microsoft's Edge.

2.4.2 Leaflet

Leaflet[3] is a very light open-source JavaScript library for creating interactive maps. Its implementation in a project does not require any credentials and world maps are available for free. Leaflet maps are fully customizable: since control elements as legends or user interaction events, to several kinds of responsive layers that can be put over the map itself.

The huge community is one of the reasons to choose it. More specifically, the community is responsible for its strongest suit: leaflet plugins. Some plugins have been indispensable for this project:

- **Leaflet-Heat**[4]. A light, simple, and fast Leaflet heatmap plugin. It uses simpleheat library under the hood, additionally clustering points into a grid for performance.

[3]`http://leafletjs.com/`
[4]By Vladimir Agafonkin (https://github.com/Leaflet/Leaflet.heat)

This heatmap is located in a Leaflet layer that is easily added or removed from the map itself

- **Leaflet-markercluster** [5]. It provides animated marker clustering functionality for Leaflet. As well as before, this applies to a Leaflet layer.

2.4.3 Sefarad

Sefarad [10] is a light environment developed to explore, analyze, and visualize data. Its main application is graphic interfaces, which is achievable due to the possibility to create complex custom dashboard with Polymer components. This Polymer instances are called widgets and they are developed separately from the dashboard, since they can be reusable. However, the fact that the widgets can be developed individually implies the necessity of a interface definition that let the dashboard communicate with the widgets.

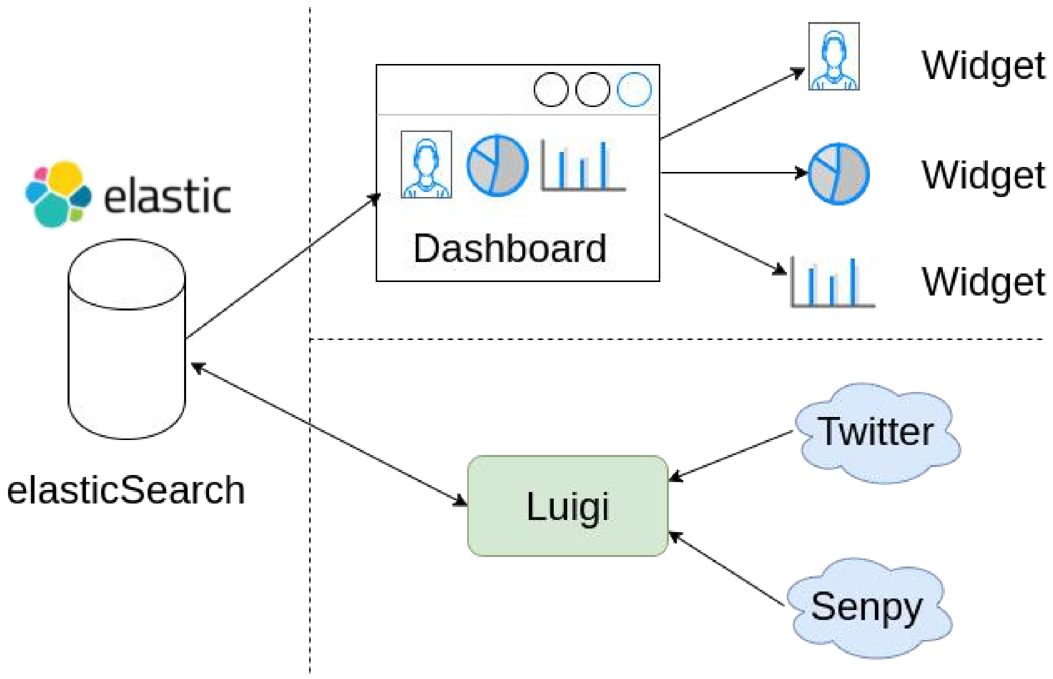


Figure 2.4: Sefarad Architechture

The main Sefarad components are:

- **Luigi**. The pipelines defined in Luigi retrieve Twitter data from Twitter API, send it to Senpy analysis endpoint, and push it to ElasticSearch.

- **ElasticSearch**. This is where Luigi pushes all data obtained to.

[5]By Dave Leaver (https://github.com/Leaflet/Leaflet.markercluster)

- **Dashboard and widgets**. The dashboard is a web instance which requests certain data to ElasticSearch and distribute them among the widgets it implements. All widgets and dashboards are made with Polymer and uses its more important features. Sefarad's principal focus is to warranty fast development of dashboards and widgets, as well as centralized visualization of data in all the ways that the widgets let. All components share data the same way: they ask for queries specifying certain filters and constraints, the dashboard itself calls the ElasticSearch client to retrieve the query data, and received data is distributed among widgets in a efficient way. Sefarad's internal data management defines makes a performance difference very important to the widget visualization fluidity.

2.5 Deployment

Nowadays, the most interesting way to deploy a service may be one that is unique, available for all operative systems, and easily maintainable. All this is possible with Docker.

2.5.1 Docker engine

Docker[3] is a software container platform characterized by the isolation it creates with the system is launched on. Unlike virtual machines, containers do not wrap a whole operating system over another, they pack libraries and settings in order to make certain service work. Besides the efficiency improvement, Docker solves lots of compatibility issues between systems, as the service will run the same regardless of where is deployed. One interesting use case would be, for example, to launch several Docker Microsoft's SQL Server instances on a AWS Linux machine.

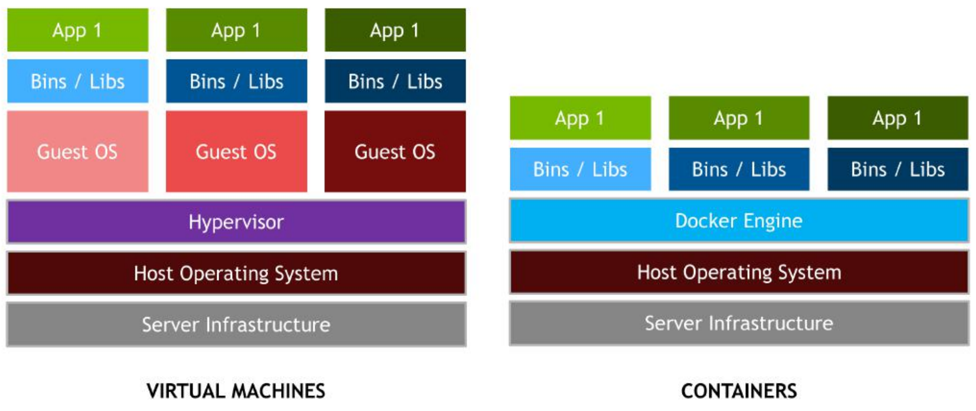


Figure 2.5: Docker VS. Virtual Machines

Hence, with a command line access in a machine and a Docker Image, it is possible to launch any kind of service as long as the necessary interfaces are mapped among host and container.

2.5.2 Docker compose

Docker Compose [2] is the CLI tool that allows to actually define and deploy custom services using the standard containerization defined by Docker. Some configuration is needed before running the command that executes the launcher:

- **Dockerfile**. Environment definition for a single container that will be deployed. This guidelines allows to install all dependencies and set working directories independently of the host operative system.

- **docker-compose.yml**. Definition of the services which the app will communicate to. All this instances will be containerized and, hence, isolated from the host. Furthermore, for each service it is possible to define disk partitions to install dependencies and save data.

  An important part of this file when working with containers behaving as a service is networks creation and assignment. A network in docker-compose allows communication among the containers, which would be impossible unless a custom routing is set in each container. Networks allows naming discovery, so it is not needed to know which are the containers IP address each time they are deployed.

  Lastly, it is also important to know that dependencies between services can be defined, so a database can be declared before a services that communicated with it.

Finally, running docker-compose up, docker-compose will manage all necessary tasks in order to have the declared services deployed with the required configuration.

2.5.3 DockerHub

As it would be expected, a technology like this has repositories where anyone can upload custom Docker Images that encapsulates different services, databases, operative systems, etc... DockerHub[4] has let this project to have images of ElasticSearch, Luigi, and Sefarad.

2.6 Bower

Bower[13] is a package and library manager for Javascript projects. It allows keeping track of all dependencies and making sure they are up to date. A Bower configuration is defined by a bower.json file, that contains the project name and all dependency names, among other information. All installed dependencies using this tool are saved under bower_components folder, which is located at the project root path. This allows to upload projects without dependencies in its content, only a bower.json is needed.

The most interesting actions that can be done with Bower are:

- **bower install [package] –save**. Install ¡package¿ under bower_components folder and saves the dependency declaration in bower.json
- **bower install bower.json**. Installs all the dependencies already defined in bower.json. This command is extremely useful when cloning a remote project in a local machine.

It is very convenient to use Bower in this project, because it integrates very well with Polymer. Besides, a massive quantity of WebComponents and Polymer elements are available via Bower.

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 what to expect from an informative dashboard.

The analysis will use the *Unified Modeling Language (UML)*[5]. This language allows us to specify, build and document a software system using graphical language. In the following UML diagrams it has been represented only the elements and attributes that concern each use case.

3.2 Use cases

This section identifies the main project's use cases. This helps to obtain the use specifications of the system, and therefore, a list of requirements for the development phase.

It is important to note that all use cases exposed below have pre-requisites. Namely, these requisites are service deployment, data fetching, data pre-processing, data analysis, data post-processing, and data indexing. Except for the deployment, all of them are encapsulated in the Luigi pipeline and explained in Section 4.2.1.

3.2.1 Use case 1: Heatmap representation

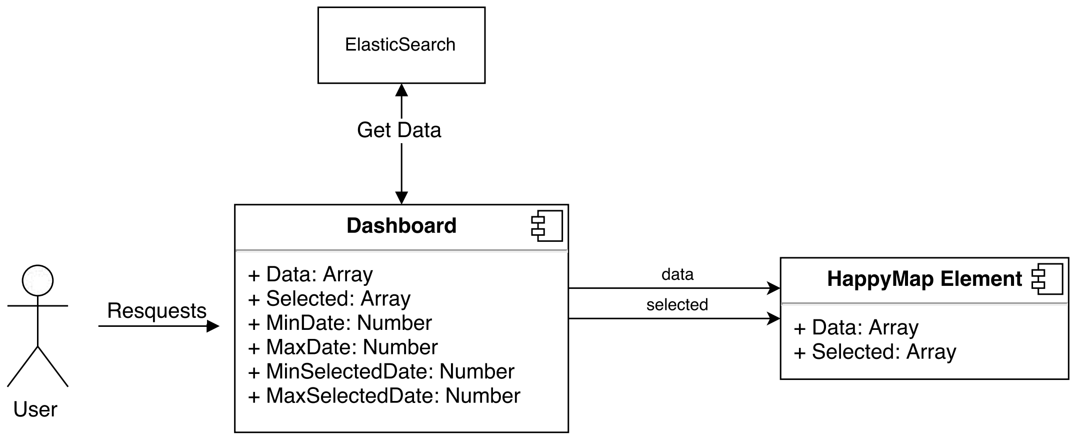


Figure 3.1: Use Case 1

- **Use Case**: Sentiment-based Heatmap representation
- **Primary Actor**: User
- **Scope**: HappyMap component
- **Description**: When dashboard is loaded, the user should see a widget that contains a map with a heatmap point representation. That points should correspond to tweets geographically located which have been analyzed by sentiment and emotion. The score on that analysis should define the intensity of each point in the heatmap. When zooming the map, the heatpoints should be recalculated relatively to the visible zone instead of remain the same.

  The map should show a legend depending on the color gradient. Besides, it also should wait to a tweet insertion in its *selected* property to create a marker layer and assign one to that tweet.

- **Basic flow**: The map should get all analyzed tweets with Polymer's property binding. Once it have them, the map should represent the coordinates saved in *coordinates* tweet property with an intensity determined by the score saved in *sentiment_analysis* tweet property. Each moment the map is zoomed in or out, all points should scale relatively to the zone displayed by spreading the points if they are separated or use an average value if they are superposing. Within map construction process, all informative labels such as the gradient legend should be created.

3.2.2 Use case 2: Tweet filtering by date

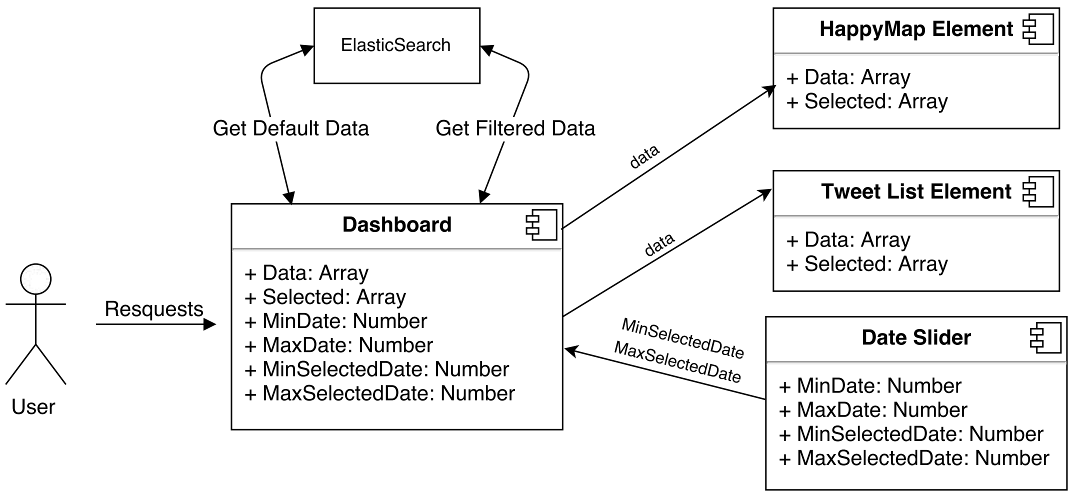


Figure 3.2: Use Case 2

- **Use Case**: Filter tweets by date
- **Primary Actor**: User
- **Scope**: Dashboard query and Date Slider component
- **Description**: In the first dashboard load, the minimum and maximum date of tweets saved in ElasticSearch have to be fetched in order to set the minimum and maximum selectable bounds for the filter. When dashboard is loaded, the user should see a clear element that suggest a interactive date filter selecting a date range. Once the user has interacted with that element, the query should automatically be requested, the dashboard data updated, and all widgets in dashboard refreshed. The selector bounds should be the same as the date property of data bounds. When some requested filter returns an empty data set, the user should be noticed.
- **Basic flow**: In the first query that the dashboard launches against ElasticSearch, two aggregates will be added to it to get minimum and maximum date of all database. Given that the user makes a modification in the selected filter range, that element should expose to its parent, the dashboard, all dates that it is representing: selected minimum date and selected maximum date are the most important ones. When the

dashboard already has the two dates, it realizes a new query with that filter, which will return a new dataset that will be passed to the *data* dashboard Polymer property. Since all widgets have to be data-bounded to the dashboard, every one of them will be updated, showing a change in its visualization.

3.2.3 Use case 3: Emotion map representation

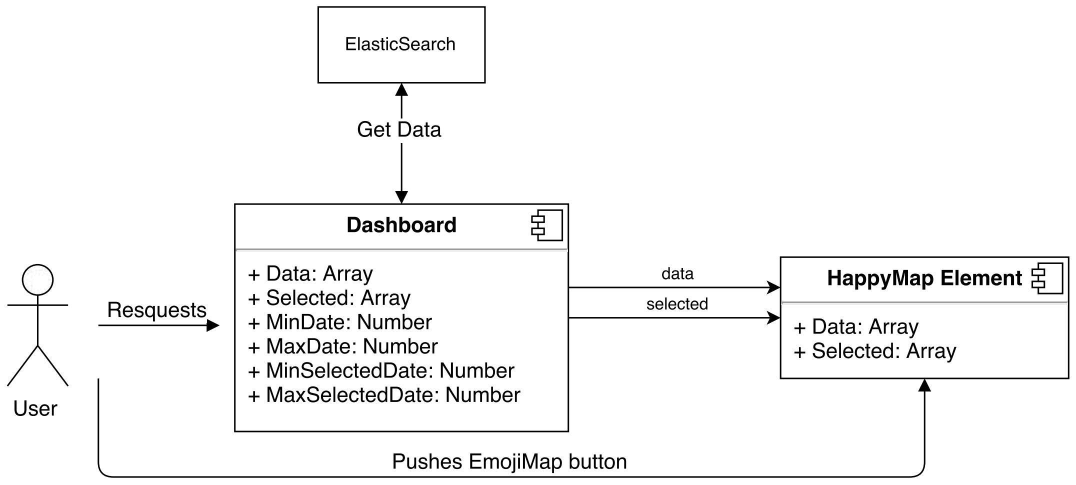


Figure 3.3: Use Case 3

- **Use Case**: Emotion-based clustered map representation
- **Primary Actor**: User
- **Scope**: HappyMap component
- **Description**: When the dashboard is loaded, there should be an option for selecting the emotion based map on HappyMap, called EmojiMap. This button should make disappear the previous visualized heatmap and display a map that bases on emotion tags obtained in the tweet analysis. Each one of that tags should be paired with an Emoji icon, which should be of the same nature of the emotion and should appear as a tweet point in the map. When lot of points appear near to each other, they should clusterize in one icon that reflects the emotion of maximum cardinality within that zone. When zooming in or out, the clusters should be recalculated and updated.
- **Basic flow**: When the EmojiMap is selected, previous layer should be removed from HappyMap. Instead of it, a new one with *Leaflet.markercluster* library should be added. One by one, all points inside *data* variable should add their coordinates and

emotion tag to the new layer. While adding that points, there should be a one to one mapping between emotion tags and Emoji images.

3.2.4 Use case 4: Tweet listing representation

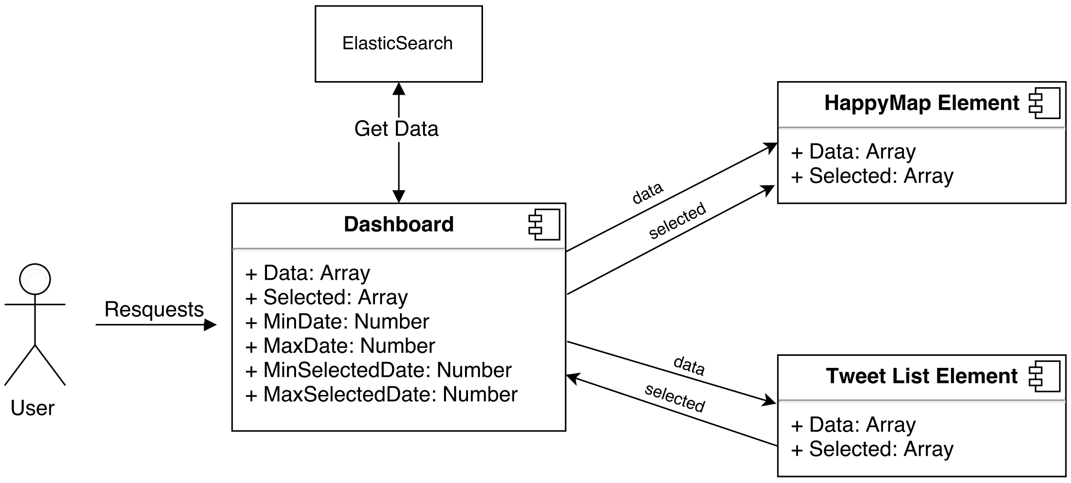


Figure 3.4: Use Case 4

- **Use Case**: Tweet list responsive representation
- **Primary Actor**: User
- **Scope**: Tweet List element and HappyMap element
- **Description**: When the dashboard is loaded, there should be an clear representation with all tweets that are being displayed in the map. This display should be in a form of a list and they should be clickable. A click in one of that list items should inform the user to the exact tweet location in HappyMap. A second click over an already selected item will remove it from the map.
- **Basic flow**: After the first query has made to ElasticSearch and a dataset have been retrieved, Tweet List element should take the very *data* Polymer property to iterate over and put some of information of them in each item. When a item is clicked, its information is saved in *selected* Polymer property, which will be read by HappyMap in order to insert markers over the map that is being displayed. A second click over an already selected item will remove it from *selected*.

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.

This project's main goal is to serve sentiment-based analyzed data into a customizable and modular dashboard. Sefarad's interfaces are followed in order to make easier the integration with already developed components and ensure compatibility with future developments.

A diagram of the general architecture is shown in Figure 4.1.

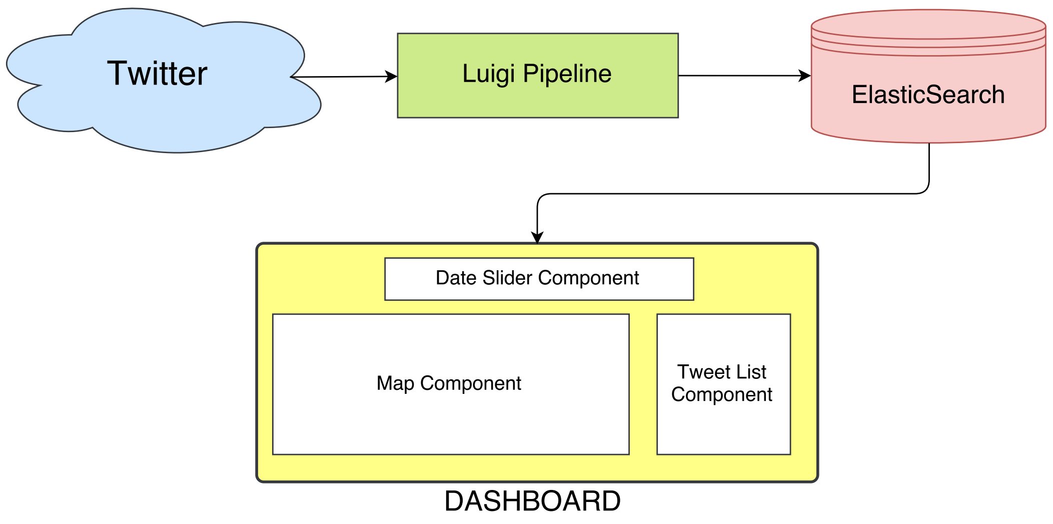


Figure 4.1: General Architecture

4.2 Modules Description

4.2.1 Luigi Pipeline - Data provision

Luigi enables the definition of pipelines, which have been used used in this project for tweet processing. Several tasks have been defined in order to provide quality information to the dashboard. At the end of the pipeline, the result is a reduced set of tweets, since several tweets that lack certain information (e.g. geo-location) are discarded.

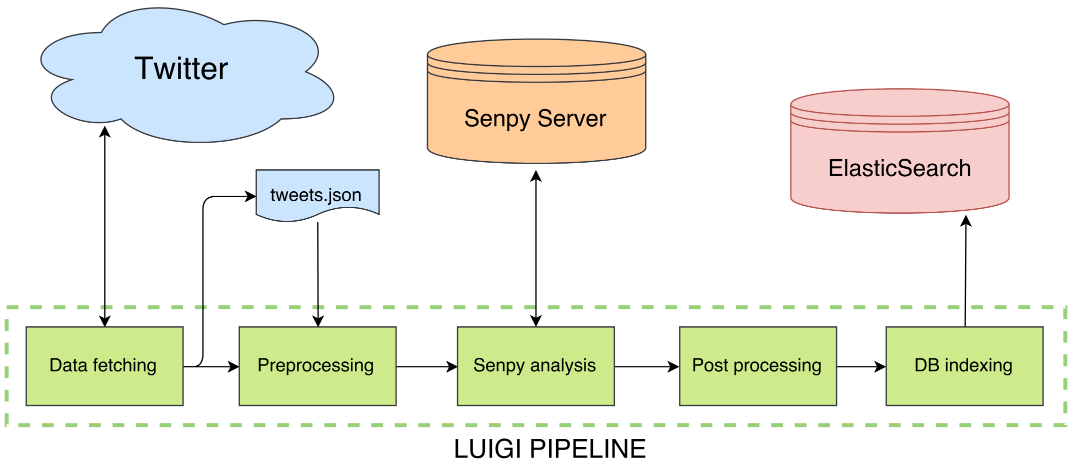


Figure 4.2: Luigi Pipeline

The pipeline defined for this project consists of several tasks:

1. **Data fetching**. As it was explained before, the social network selected for this data processing has been Twitter. Thus, in order to obtain tweets, it is needed a communication with the servers, via REST API. Twitter offers the Twitter API [6], which can retrieve almost every type of information hosted publicly by the organization. In this case, the only wanted entities are Tweet Objects, modeled as JSONs with lots of properties.

   Therefore, in this pipeline task, thousands of tweets between two dates are got using the Python Twitter Client. There are not further filters applied to this query. Once all tweets have been fetched, they are saved in a JSON lines files (2.3.2).

   It is important to mention that, if this generated JSON file already exists, luigi will skip this step, making a faster pipeline procesing.

2. **Pre-processing**. This step is necessary for avoiding unuseful tweets taking space in the ElasticSearch database. A selection of the fetched dataset is made following certain criteria.

   The most important step is to ensure the current processing tweet has enough geolocation information. As Twitter App allows to disable tweet geolocation, only approximately 15% of tweets have accurate coordinates. However, it is possible to calculate a non accurate position that can suite this case. Thanks to the *Places*[1] key present in the tweet JSON object, a zone from which a tweet could be posted is defined. Actually, this object contains latitude and longitude values for vertices that correspond to a polygon which delimit the space range guessed by Twitter. Therefore, the medium point for that coordinates is calculated and used as true geolocation for the tweet.

   Finally, this step also includes restrictions to ensure that the results in the dashboard will be meaningful, such as ensuring tweet dates are separated in time or space.

3. **Senpy Analysis**. After the previous step, the pipeline takes advantage of Senpy's REST API. Emotion-anew, the plugin used in Senpy for emotion and sentiment analysis, only requires the message string, so a POST call is sent towards Senpy's endpoint with the tweet message in the payload. In order to send that, Python's Requests library is used.

   Senpy response is a JSON containing valence, arousal and dominance punctuation, as well as a tag for emotion among joy, sadness, negative-fear, disgust, anger and neutral. The current task takes that information and pushes them into the tweet object as value for "sentiment_analysis" field.

[1]In Twitter docs: *Tweets associated with places are not necessarily issued from that location but could also potentially be about that location.* (https://dev.twitter.com/overview/api/places)

4. **Post-processing**. Now that the current tweet has been analyzed, a filter can also be applied to ensure the analysis quality. The biggest disturbance added in previous task is the quantity of results tagged as neutral. Neutral tweets could reach the 85% of the dataset, so in this step they are discarded for graphic representation simplicity in the dashboard. Obviously, there are lot of information lost in this step, but is important to understand that this decision is only made because of the nature of this project and it should not be done in a professional environment. Due to this decision, the future emotional representation will not be dominated deterministically by any tag.

5. **DB indexing**. Like its name reads, this task mission is to push into the ElasticSearch the already selected tweets, which complies with all the restrictions explained before. Owing to Luigi's integration with ElasticSearch, it is easy to finish all this flow with an indexation. Few simple code lines tells Luigi which is the DB's IP address, and what index it must post the tweet into.

Finally, after all this steps, Luigi has ended its work in this project and all the data is provisioned.

4.2.2 ElasticSearch Index

Elasticearch is organized in documents contained within indexes (refsec:elasticsearch). Selecting a structure of indexes and a well-oriented mapping for the documents is crucial for performance. However, the use cases in this project do not require such tuning. Thus, Elasticsearch will only have one index, called *tweets_happymap*, and there will not be a fixed mapping for it. All data provisioned will be only posted once and we assumed that data will only be stored via the pipeline.

4.2.3 Dashboard

The dashboard is, as the widgets it covers, a Polymer component. Its main purpose is to fetch tweets from ElasticSearch and to distribute them to each widget that demands them. This is possible due to the inclusion of certain Polymer properties into the dashboard, which makes transparent the data flow between components.

This dashboard counts with properties named below:

- **Data**. The most important property and the only that is compulsory independently of the widgets included. It is an array, and it contains the result of the last ElasticSearch

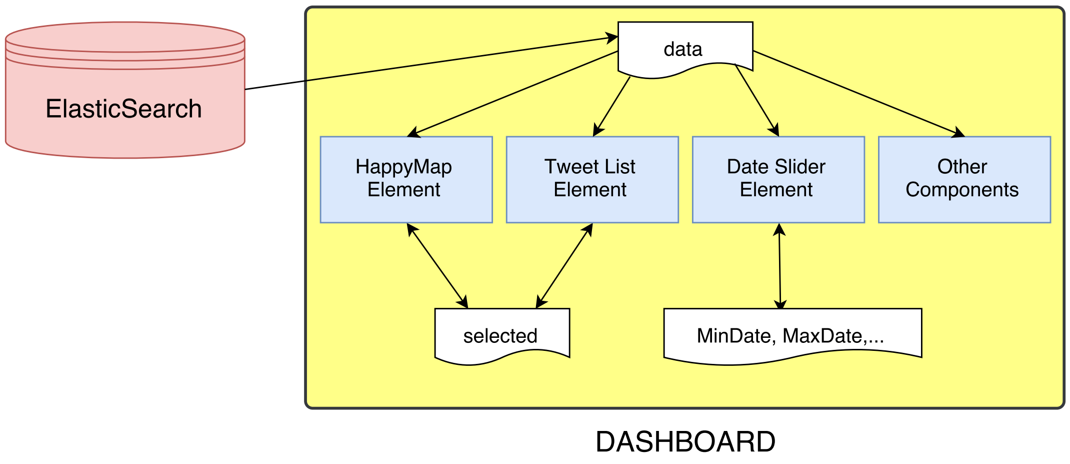


Figure 4.3: Dashboard Data Flow

query. With this project's widget setup, data only flows downwards, in other words, data is never changed by the widgets, but all widgets receive data. This is not the only way to establish the data flow, but it is for sure the safest for stability.

- **Selected**. This property is a dependency fixed by Happymap Component and Tweet List Component. It is, as the previous property, an array that saves tweets. However, in selected there will only be those tweets selected by the user in interaction with the UI. Selected allows a interaction between the tweet selection in Tweet List Element and the appearance of a pin in HappyMap Element.

- **MinDate, MaxDate, DefaultSelectedMin, DefaultSelectedMax, Selected-Min, SelectedMax**. All this properties are dependencies fixed by Date Slider Element, and they concern the position of the bar that defines the time filter set to the ElasticSearch query. This will be better explained in the next section.

4.3 Dashboard Polymer Components

In this section, each one of the developed Polymer components will be explained with detail. Note that this components are not the only ones that can be added to the dashboard, since the modularity actually allows the inclusion of new ones very easily.

4.3.1 HappyMap Element

HappyMap Element is the core widget in the dashboard, as it is the only one that actually represents the sentiment analysis applied to the tweets got from Twitter.

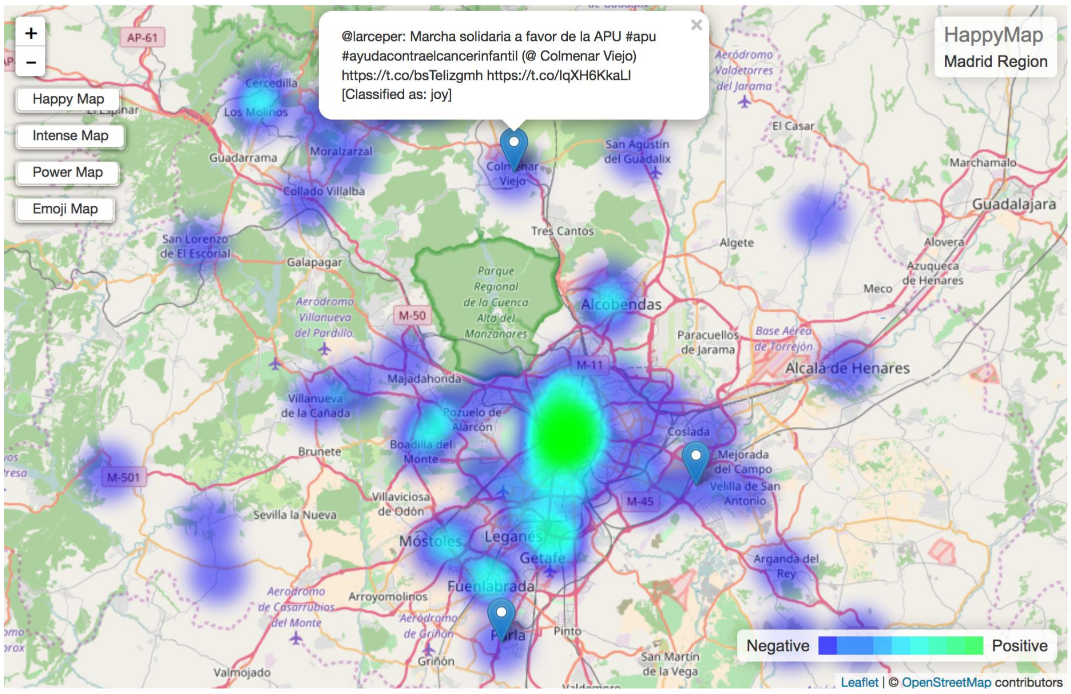

@larceper: Marcha solidaria a favor de la APU #apu #ayudacontraelcancerinfantil (@ Colmenar Viejo) https://t.co/bsTelizgmh https://t.co/IqXH6KkaLI [Classified as: joy]

Figure 4.4: HappyMap Snapshot - HappyMap Mode

This Polymer component is composed visually only by a Leaflet map with some elements above it:

- **Information tag**: Shows which of the layers is being displayed at the moment. Its location is up-right.
- **Heatmap legend**: In case a heatmap is active, a color leyend shows whether each color means a positive or negative value of the correspondent VAD value. It is located down-right.
- **Switch buttons**: Some buttons are placed at the map's left side in order to toggle the layer currently being displayed.
- **Layers**: They are superposed to the map and there is only one active at the same time. Each layer will be described later.

HappyMap counts with two of the previously explained properties: data and selected. They are used, as it was exposed, for defining tweets inputs for the map. Within HappyMap's logic, all dataset got is iterated, retrieving geographic coordinates, emotion tag, tweet text, tweet user, and score for the VAD values. Every one of these available variables are used for the active layer representation, which can be classified in two categories: heatmap layers and clusterization layers.

4.3.1.1 Heatmap Layers

A heatmap layer is adequate to represent the variations of a numeric value over a geographic zone. This layer's purpose is to provide flashy information to the user, taking advantage of attractive color gradients. In this case, the project focus has been centred into representing Valence, Arousal, and Dominance values over the map. In order to clearly differentiate each one, different colors have been chosen.

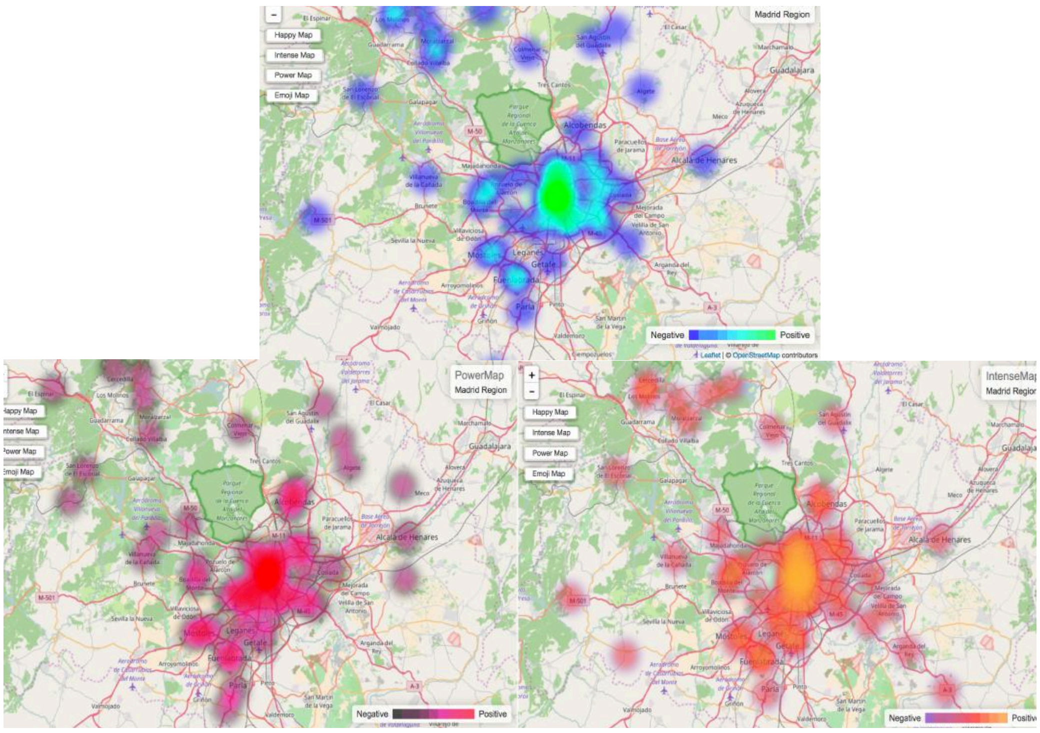


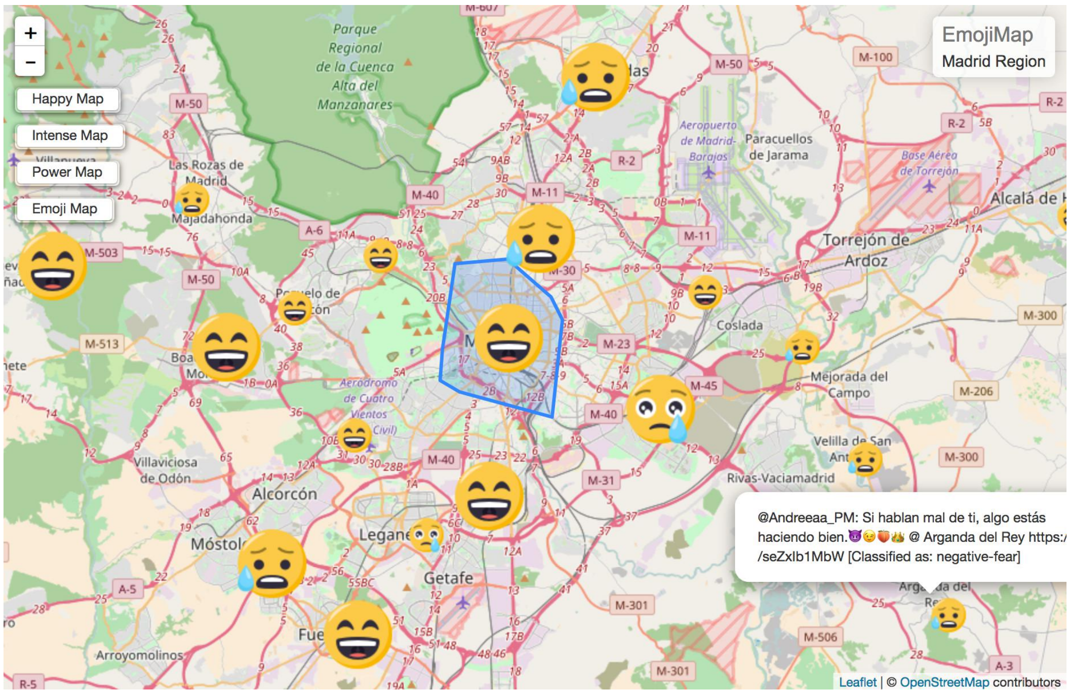
Figure 4.5: HappyMap (up), PowerMap (left), and IntenseMap (right)

- **HappyMap**: Representation of the Valence value, which means the pleasantness evoked by the text tweet. The chosen gradient comes from negative blue to positive lime.
- **PowerMap**: Representation of the Dominance value, which means the control degree felt over the text tweet. The chosen gradient comes from negative black to positive passion red.
- **IntenseMap**: Representation of the Arousal value, which means the intensity of emotion provoked by the text tweet. The chosen gradient comes from negative purple to positive orange.

When any of that three layers are selected, legend and info tag are refreshed and changed accordingly.

4.3.1.2 Clustering Layer

As last named layers focused over VAD values representation, with this clustering layer the objective is to give a general emotion overview over the zone. This layer is called EmojiMap, and the value it represents is each tweet's emotion tag retrieved in the sentiment analysis.



@Andreeaa_PM: Si hablan mal de ti, algo estás haciendo bien.😈😉🍑👑 @ Arganda del Rey https:/ /seZxlb1MbW [Classified as: negative-fear]

Figure 4.6: EmojiMap Snapshot

When activating the EmojiMap and depending on the dataset size and tweets position, the map becomes divided into polygonal zones. These zones inform users of the geographic coverage of a certain cluster point, and they are generated joining all coordinates of tweets bounded by a circle with center in the cluster point and constant pre-defined radio. The vertices of the polygon are the furthermost points to the center and it is displayed in blue (visible while hovering the mouse over it).

There are several tweets contained withing the zone, and the emotion of the majority of them defines the emoji shown over the polygon. If the icon is clicked or scroll is used, the map will zoom, the polygons will be recalculated over the new part of the map displayed, and the emojis reset. Besides, the icon changes are smoothly animated with a nice fading.

Furthermore, it is noticeable that there are big and small face icons. This is because they have two meanings: the big ones represent a tweet cluster, and the small ones represent a only tweet. In the last case, the the click event over the emoji pops up tweet information as tweet text, tweet user, and emotion tag name.

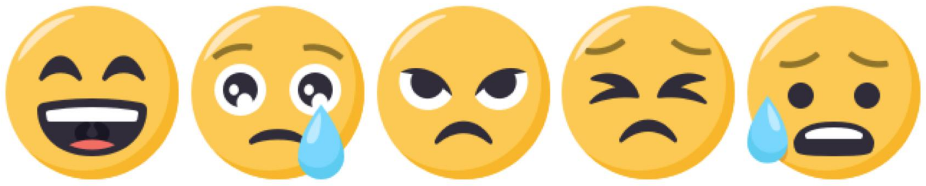


Figure 4.7: From left to right, respectively: joy, sadness, anger, disgust, negative-fear

Next Figure represents each one of the emojis available in this layer. All of them have been chosen from a open-source provider [2].

The source code of HappyMap is available as a public git repository[3].

4.3.2 Tweet List Element

Tweet List is a Polymer component whose purpose is to facilitate the representation of any tweet quantity, and the possibility of tweet selection by the user. It is visually composed by iterative Polymer directives and Bootstrap styled list items. From the logic point of view, it only implement data and selected Polymer properties, which already have been named many times.

This element is the only one in this project's dashboard setting that can add tweet to the selected variable, via the *onClick* event in list items. Since Polymer data binding enables automatic data flows between components, the dashboard and map are aware of this changes and react conveniently about it. Nevertheless, Tweet List also reacts automatically to data property changes, to reflect the queries requested from the dashboard.

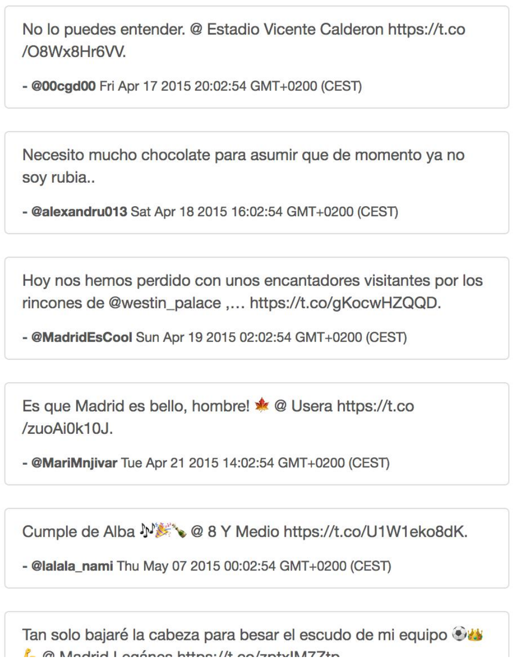
Each one of the list items displayed are fulfilled with tweet text, user, and timestamp.

Tweet List is available as a public git repository[4].

[2]https://www.emojione.com

[3]`https://lab.cluster.gsi.dit.upm.es/sefarad/happymap`

[4]`https://lab.cluster.gsi.dit.upm.es/sefarad/tweet-list`


No lo puedes entender. @ Estadio Vicente Calderon https://t.co/O8Wx8Hr6VV.

- @00cgd00 Fri Apr 17 2015 20:02:54 GMT+0200 (CEST)

Necesito mucho chocolate para asumir que de momento ya no soy rubia..

- @alexandru013 Sat Apr 18 2015 16:02:54 GMT+0200 (CEST)

Hoy nos hemos perdido con unos encantadores visitantes por los rincones de @westin_palace ,... https://t.co/gKocwHZQQD.

- @MadridEsCool Sun Apr 19 2015 02:02:54 GMT+0200 (CEST)

Es que Madrid es bello, hombre! 🍁 @ Usera https://t.co/zuoAi0k10J.

- @MariMnjivar Tue Apr 21 2015 14:02:54 GMT+0200 (CEST)

Cumple de Alba 🎶🎉🍾 @ 8 Y Medio https://t.co/U1W1eko8dK.

- @lalala_nami Thu May 07 2015 00:02:54 GMT+0200 (CEST)

Tan solo bajaré la cabeza para besar el escudo de mi equipo ⚽👑

Figure 4.8: Tweet List Element Snapshot

4.3.3 Date Slider Element

Date Slider is a Polymer component that allows to select date ranges between two bounding dates. These dates, in this project's case, are used to tell the dashboard which limits has to establish in its query to ElasticSearch. Of course, in other application, as modularity is guaranteed, that dates could be used with other goals.

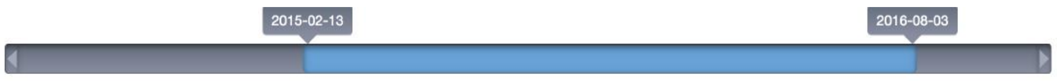


Figure 4.9: Date Slider Element Snapshot

Visually, this component have taken advantage from a open-souce jquery library called jQRangeSlider [1], which provides a customizable slide-able bar that accepts Date Javascript objects as values. The selected date range bounds are always displayed in order to be easier to understand by the user.

From the logical point of view, Date Slider has several Polymer properties that let the dashboard know what the selected range is, and the user which are their decision capability over the filter.

- **Min, Max**: Absolute bounds of the slider, in other words, selection limits for the user when moving the bar. This properties are set by the dashboard when, with the ElasticSearch's "aggs" response, minimum and maximum tweet timestamp are retrieved. Min and Max properties will never change unless the dataset indexed in ElasticSearch has been modified.

- **DefaultSelectedMin, DefaultSelectedMin**: Default bounds for the slider selected zone. They are also provided in first instance by the dashboard, since they correspond to the limit timestamps of the first not-filtered query.
- **SelectedMin, SelectedMax**: This are the bounds of the slider selected zone chosen by the user. Until the user interacts with Date Slider, they have default values. Once the bar has moved, the current selected bounds are stored in this properties.

All Date Slider properties are modelled as integers and they represent millisecond timestamps since epoch [5]. The reason for this is that is easier for the dashboard to use a millisecond mark to filter its queries to ElasticSearch. Furthermore, the aggregations to obtain maximum and minimum date are lot simpler. Of course, although all date management is with millisecond timestamps, each UI date representation is preceded by a transformation to a Date Javascript object.

Date Slider is available as a public git repository[6].

[5]The time kept internally by a computer system is usually expressed as the number of time units that have elapsed since a specified date. Unix-like systems and most programming languages takes January 1, 1970

[6]`https://lab.cluster.gsi.dit.upm.es/sefarad/date-slider`

CHAPTER 5

Case study

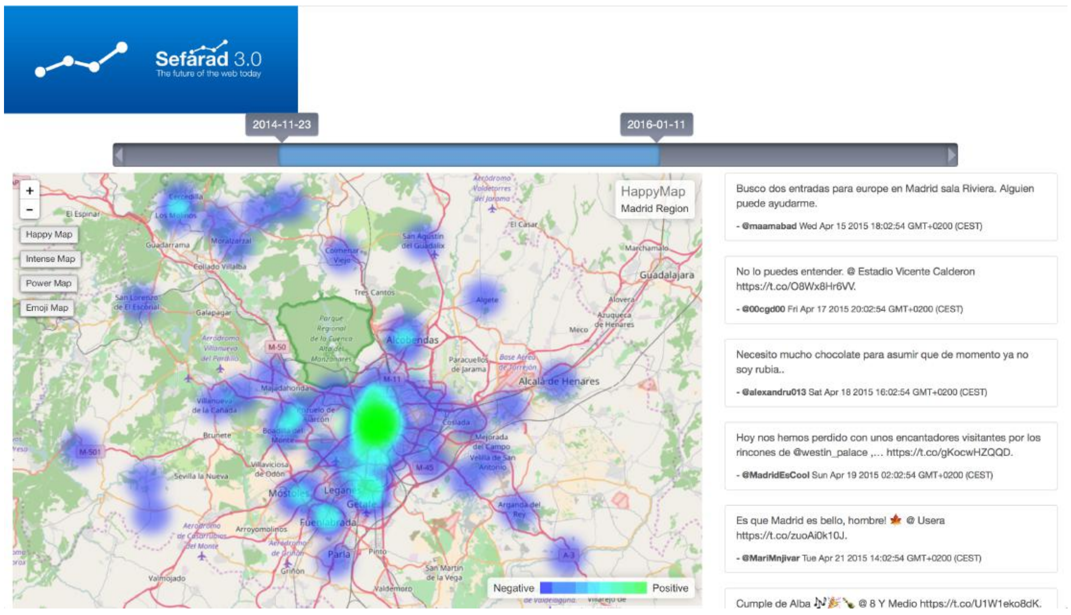
5.1 Introduction

In this chapter we are going to describe a normal use of the dashboard, and cover all use cases defined in 3.2. Its main purpose is to completely understand the dashboard functionalities. From now on, it is supposed that all deployments and data provisioning have been executed and properly succeeded.

5.2 Use case 1: Heatmap representation

First, the user has to enter in any modern internet browser the assigned dashboard endpoint in port 8080 (in case of the provided demo, the endpoint is *localhost:8080*). The dashboard, as reflected in Figure 3.1, firstly get a set of tweets from Elasticsearch. Specifically, it makes a *match_all* query with a limit of 100 hits, for demo simplicity sake. Once this data is got, it is assigned to *data* Polymer property, which will be passed along to HappyMap element (and all other widgets). The heatmap boots with a non empty *data* field, what implies that will be content displayed in the map at first instance.

Figure 5.1 shows the initial screen provided by the service.



Busco dos entradas para europe en Madrid sala Riviera. Alguien puede ayudarme.

- **@maamabad** Wed Apr 15 2015 18:02:54 GMT+0200 (CEST)

No lo puedes entender. @ Estadio Vicente Calderon https://t.co/O8Wx8Hr6VV.

- **@00cgd00** Fri Apr 17 2015 20:02:54 GMT+0200 (CEST)

Necesito mucho chocolate para asumir que de momento ya no soy rubia..

- **@alexandru013** Sat Apr 18 2015 16:02:54 GMT+0200 (CEST)

Hoy nos hemos perdido con unos encantadores visitantes por los rincones de @westin_palace ,... https://t.co/gKocwHZQQD.

- **@MadridEsCool** Sun Apr 19 2015 02:02:54 GMT+0200 (CEST)

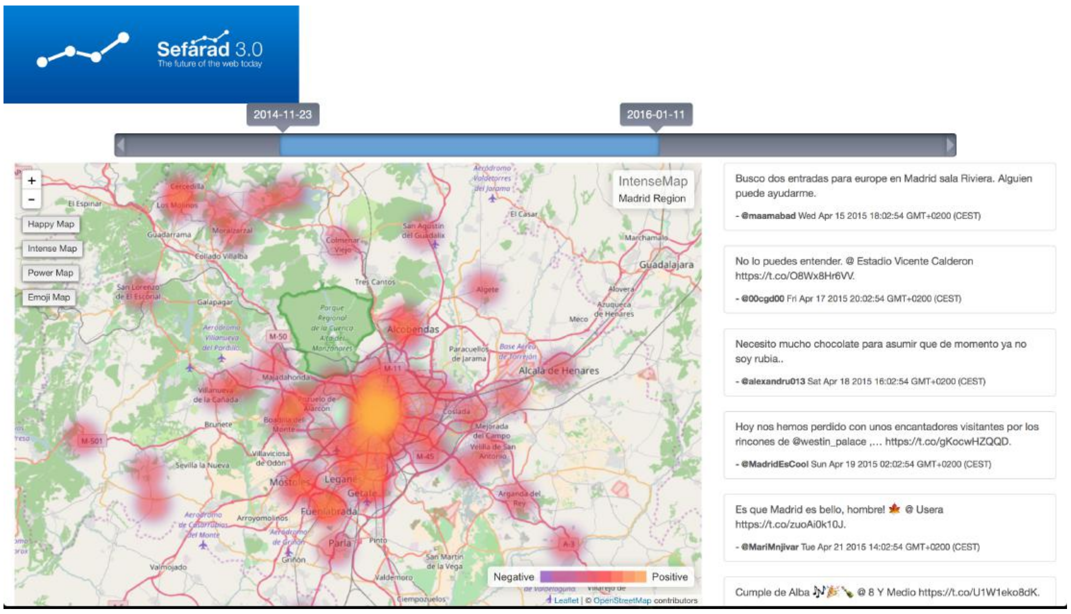
Es que Madrid es bello, hombre! 🍁 @ Usera https://t.co/zuoAi0k10J.

- **@MariMnjivar** Tue Apr 21 2015 14:02:54 GMT+0200 (CEST)

Cumple de Alba 🎶🎉🍾 @ 8 Y Medio https://t.co/U1W1eko8dK.

Figure 5.1: Dashboard displaying Happymap

Default layer displayed in the map is HappyMap. The positiveness (valence value) of each tweet is what is being represented by the color gradient. At this point, the user can interact with any of the three widgets. For example, they could click the switching map buttons at the left of the map, what would change the analysis tweet value to arousal or dominance. What the map component is doing is setting all layers but the selected by the button as detached from the actual Leaflet map. If there are any markers in the map, they will not be erased, in order to maintain user selection if they want to make comparisons. Figures 5.2 and 5.3 are the results of pushing IntenseMap and PowerMap buttons respectively.



Busco dos entradas para europe en Madrid sala Riviera. Alguien puede ayudarme.

- **@maamabad** Wed Apr 15 2015 18:02:54 GMT+0200 (CEST)

No lo puedes entender. @ Estadio Vicente Calderon https://t.co/O8Wx8Hr6VV.

- **@00cgd00** Fri Apr 17 2015 20:02:54 GMT+0200 (CEST)

Necesito mucho chocolate para asumir que de momento ya no soy rubia..

- **@alexandru013** Sat Apr 18 2015 16:02:54 GMT+0200 (CEST)

Hoy nos hemos perdido con unos encantadores visitantes por los rincones de @westin_palace ,... https://t.co/gKocwHZQQD.

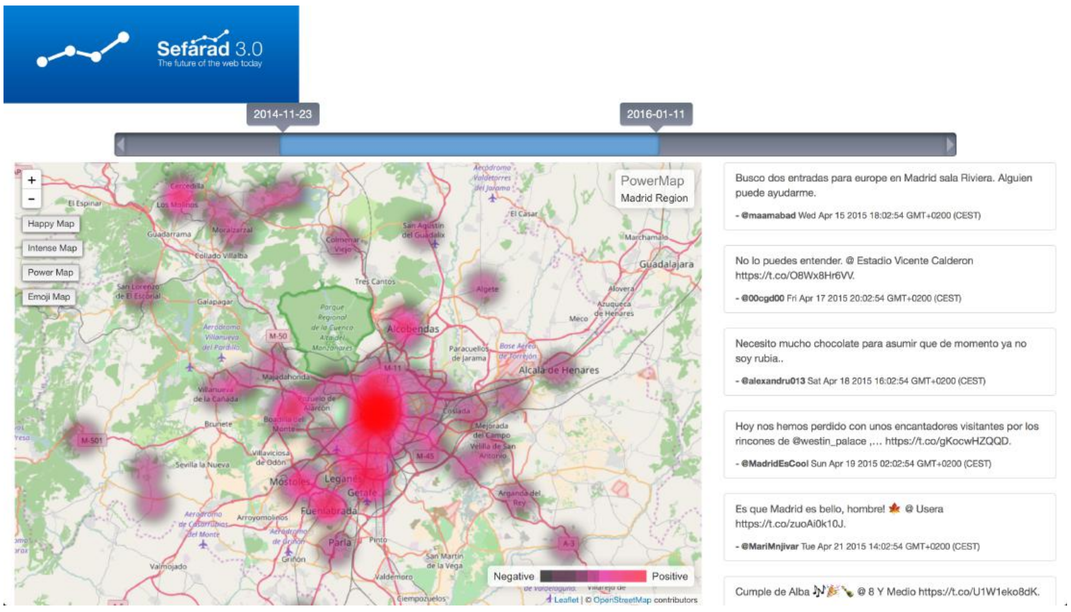
- **@MadridEsCool** Sun Apr 19 2015 02:02:54 GMT+0200 (CEST)

Es que Madrid es bello, hombre! 🍁 @ Usera https://t.co/zuoAi0k10J.

- **@MariMnjivar** Tue Apr 21 2015 14:02:54 GMT+0200 (CEST)

Cumple de Alba 🎶🎉🍾 @ 8 Y Medio https://t.co/U1W1eko8dK.

Figure 5.2: Dashboard displaying IntenseMap



Busco dos entradas para europe en Madrid sala Riviera. Alguien puede ayudarme.

- **@maamabad** Wed Apr 15 2015 18:02:54 GMT+0200 (CEST)

No lo puedes entender. @ Estadio Vicente Calderon https://t.co/O8Wx8Hr6VV.

- **@00cgd00** Fri Apr 17 2015 20:02:54 GMT+0200 (CEST)

Necesito mucho chocolate para asumir que de momento ya no soy rubia..

- **@alexandru013** Sat Apr 18 2015 16:02:54 GMT+0200 (CEST)

Hoy nos hemos perdido con unos encantadores visitantes por los rincones de @westin_palace ,... https://t.co/gKocwHZQQD.

- **@MadridEsCool** Sun Apr 19 2015 02:02:54 GMT+0200 (CEST)

Es que Madrid es bello, hombre! 🍁 @ Usera https://t.co/zuoAi0k10J.

- **@MariMnjivar** Tue Apr 21 2015 14:02:54 GMT+0200 (CEST)

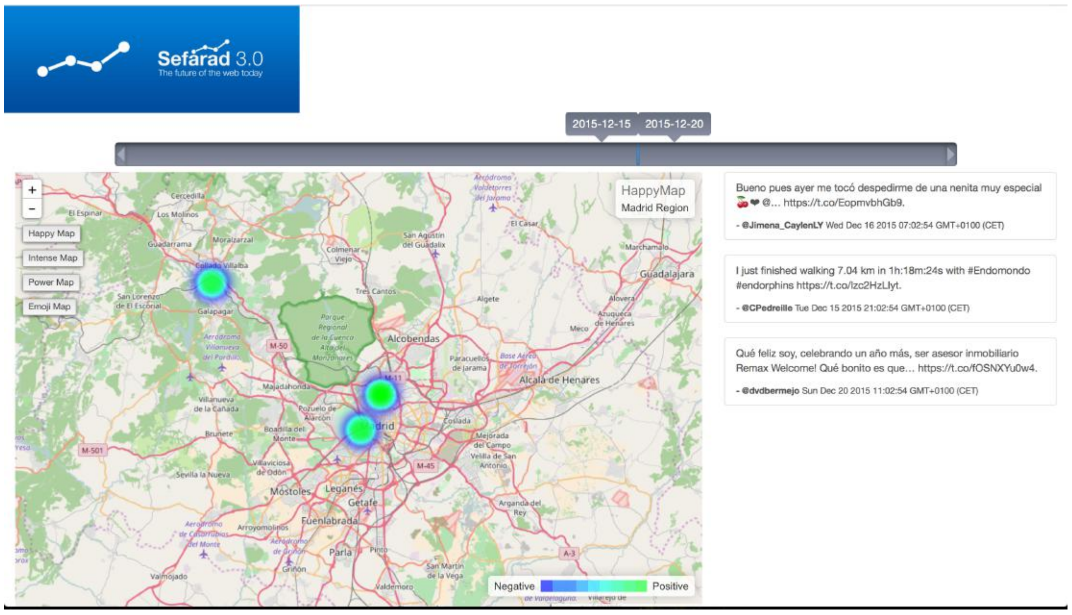
Cumple de Alba 🎶🎉🍾 @ 8 Y Medio https://t.co/U1W1eko8dK.

Figure 5.3: Dashboard displaying PowerMap

5.3 Use case 2: Tweet filtering by date

The default query made by the dashboard will get data with no criteria, however this use case lets apply some logic to that request. Once all dashboard is loaded and widgets are actively showing its UI, the user could resize and move Date Slider selection bar. This overrides *data* Polymer propery saving the new dataset retrieved from ElasticSearch. As explained before, *data* is transferred to all widgets thanks to Polymer's property binding, thus all other components will be updated in a very short (almost inappreciable) amount of time. From a user point of view, when moving Data Slider, all components react accordingly in order to display the data withing the dates set by them.

Figure 5.4 is an example selection of a very low date range.



Bueno pues ayer me tocó despedirme de una nenita muy especial 🍒❤ @... https://t.co/EopmvbhGb9.

- @Jimena_CaylenLY Wed Dec 16 2015 07:02:54 GMT+0100 (CET)

I just finished walking 7.04 km in 1h:18m:24s with #Endomondo #endorphins https://t.co/lzc2HzLlyt.

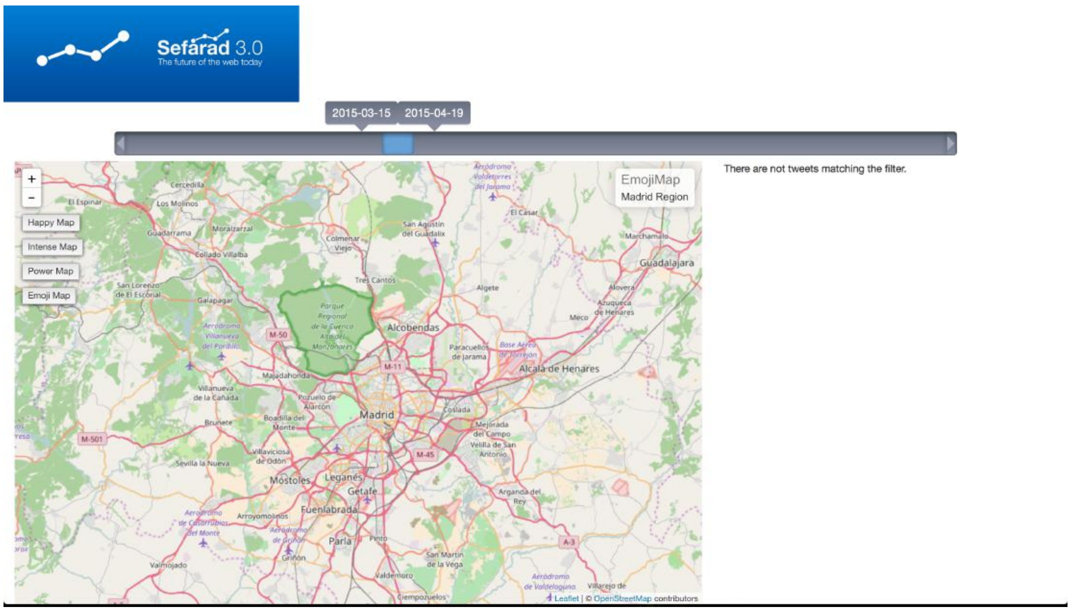
- @CPedreille Tue Dec 15 2015 21:02:54 GMT+0100 (CET)

Qué feliz soy, celebrando un año más, ser asesor inmobiliario Remax Welcome! Qué bonito es que... https://t.co/fOSNXYu0w4.

- @dvdbermejo Sun Dec 20 2015 11:02:54 GMT+0100 (CET)

Figure 5.4: Date Slider selecting few tweets

If this selection defines a date range that has no response from the database, no points will be displayed in the map and an informative message will be shown by Tweet List, as in Figure 5.5.



There are not tweets matching the filter.

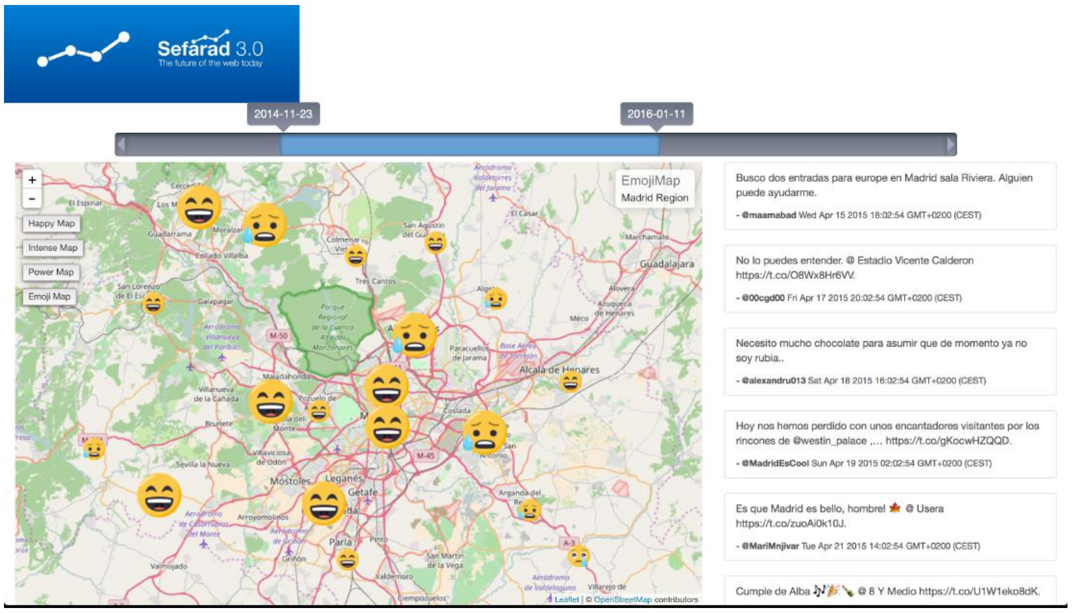
Figure 5.5: Date Slider not selecting any tweets

5.4 Use case 3: Emotion map representation

In HappyMap element, all layers are created and fed with data, it is the user interaction with buttons provided what determines if a layer become active or not. Therefore, EmojiMap, the map for emotional representation, is already there but non-visible. In case that the user would want to know how emotions are classified in the current geographic zone, they should push the EmojiMap button. All heatmaps will be detached from Leaflet map, EmojiMap will become active, and Figure 5.6 would be displayed.

The clusterization of all points displayed will vary from query to query, as each point likely has a unique geographical location within the dataset. This means that, if EmojiMap is activated and Date Slider is used to modify the ElasticSearch query, all icons will change in order to reflect which are the emotion tags of the new dataset.

Furthermore, the user could want to get a more geographically accurate information about one of the values represented in the heatmaps. Then, they can zoom in the map, which would make the map recalculate relatively all heatmap points, showing a more defined representation of each tweet. If zoom is applied into Figure 5.8, what would be got is Figure 5.7



Busco dos entradas para europe en Madrid sala Riviera. Alguien puede ayudarme.

- **@maamabad** Wed Apr 15 2015 18:02:54 GMT+0200 (CEST)

No lo puedes entender. @ Estadio Vicente Calderon https://t.co/O8Wx8Hr6VV.

- **@00cgd00** Fri Apr 17 2015 20:02:54 GMT+0200 (CEST)

Necesito mucho chocolate para asumir que de momento ya no soy rubia..

- **@alexandru013** Sat Apr 18 2015 16:02:54 GMT+0200 (CEST)

Hoy nos hemos perdido con unos encantadores visitantes por los rincones de @westin_palace ,... https://t.co/gKocwHZQQD.

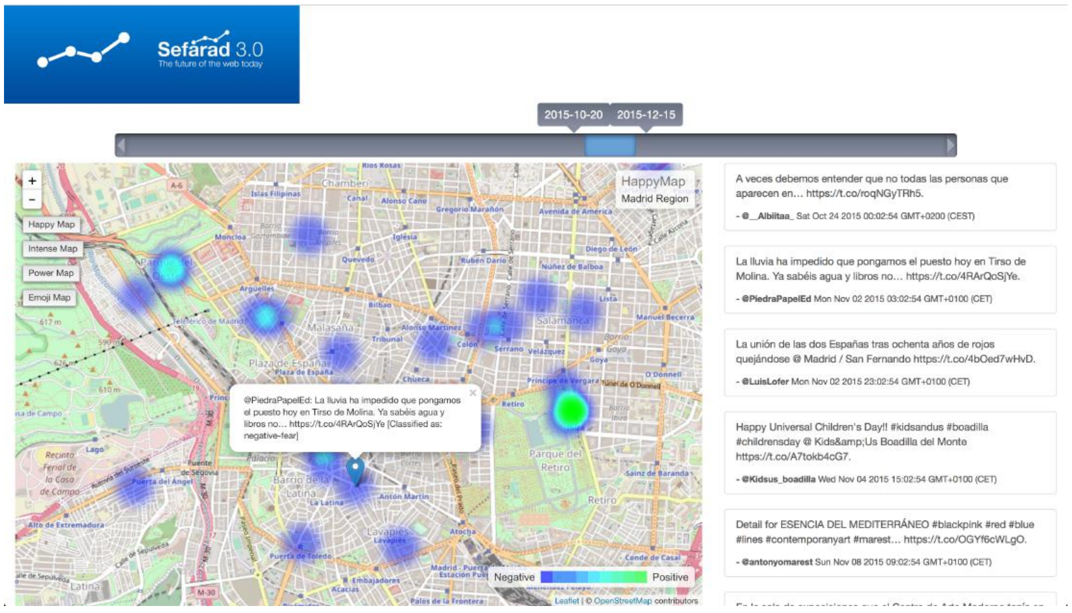
- **@MadridEsCool** Sun Apr 19 2015 02:02:54 GMT+0200 (CEST)

Es que Madrid es bello, hombre! 🍁 @ Usera https://t.co/zuoAi0k10J.

- **@MariMnjivar** Tue Apr 21 2015 14:02:54 GMT+0200 (CEST)

Cumple de Alba 🎶🎉🍾 @ 8 Y Medio https://t.co/U1W1eko8dK.

Figure 5.6: Dashboard displaying EmojiMap



@PiedraPapelEd: La lluvia ha impedido que pongamos el puesto hoy en Tirso de Molina. Ya sabéis agua y libros no... https://t.co/4RArQoSjYe [Classified as: negative-fear]

A veces debemos entender que no todas las personas que aparecen en... https://t.co/roqNGyTRh5.

- **@__Albiitaa_** Sat Oct 24 2015 00:02:54 GMT+0200 (CEST)

La lluvia ha impedido que pongamos el puesto hoy en Tirso de Molina. Ya sabéis agua y libros no... https://t.co/4RArQoSjYe.

- **@PiedraPapelEd** Mon Nov 02 2015 03:02:54 GMT+0100 (CET)

La unión de las dos Españas tras ochenta años de rojos quejándose @ Madrid / San Fernando https://t.co/4bOed7wHvD.

- **@LuisLofer** Mon Nov 02 2015 23:02:54 GMT+0100 (CET)

Happy Universal Children's Day!! #kidsandus #boadilla #childrensday @ Kids&amp;Us Boadilla del Monte https://t.co/A7tokb4cG7.

- **@Kidsus_boadilla** Wed Nov 04 2015 15:02:54 GMT+0100 (CET)

Detail for ESENCIA DEL MEDITERRÁNEO #blackpink #red #blue #lines #contemporanyart #marest... https://t.co/OGYf6cWLgO.

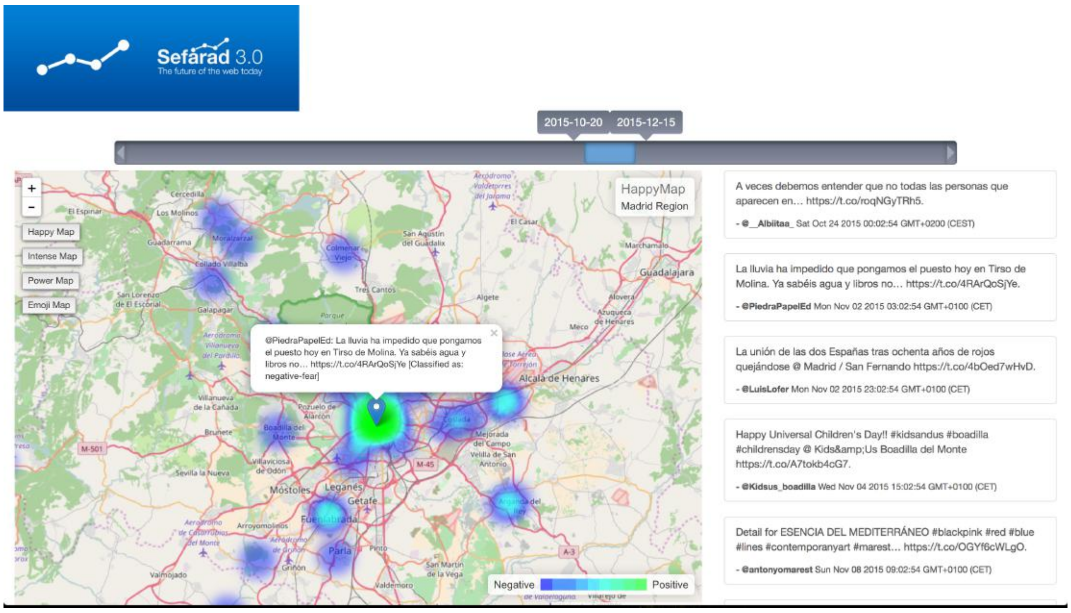
- **@antonyomarest** Sun Nov 08 2015 09:02:54 GMT+0100 (CET)

Figure 5.7: Heatmap points recalculation when zooming

5.5 Use case 4: Tweet listing representation

In a normal scenario in which there are several points represented in any of the map, the user could click in any of the Tweet List items to get a marker in HappyMap. This would show visually to the user the exact location of this tweet. Besides, that marker is also clickable, popping up message information. This is shown in Figure 5.8.

When clicking, Tweet List will simply add the selected tweet to *selected* Polymer property, which automatically will be passed to the dashboard and, then, to HappyMap element or others that would use the that array.



@PiedraPapelEd: La lluvia ha impedido que pongamos el puesto hoy en Tirso de Molina. Ya sabéis agua y libros no… https://t.co/4RArQoSjYe [Classified as: negative-fear]

A veces debemos entender que no todas las personas que aparecen en… https://t.co/roqNGyTRh5.

- **@__Albiitaa_** Sat Oct 24 2015 00:02:54 GMT+0200 (CEST)

La lluvia ha impedido que pongamos el puesto hoy en Tirso de Molina. Ya sabéis agua y libros no… https://t.co/4RArQoSjYe.

- **@PiedraPapelEd** Mon Nov 02 2015 03:02:54 GMT+0100 (CET)

La unión de las dos Españas tras ochenta años de rojos quejándose @ Madrid / San Fernando https://t.co/4bOed7wHvD.

- **@LuisLofer** Mon Nov 02 2015 23:02:54 GMT+0100 (CET)

Happy Universal Children's Day!! #kidsandus #boadilla #childrensday @ Kids&amp;Us Boadilla del Monte https://t.co/A7tokb4cG7.

- **@Kidsus_boadilla** Wed Nov 04 2015 15:02:54 GMT+0100 (CET)

Detail for ESENCIA DEL MEDITERRÁNEO #blackpink #red #blue #lines #contemporanyart #marest… https://t.co/OGYf6cWLgO.

- **@antonyomarest** Sun Nov 08 2015 09:02:54 GMT+0100 (CET)

Figure 5.8: Tweet selected and marker clicked

All this dashboard integration and a simple demo can be checked at *https://lab.cluster.gsi.dit.upm.es/sef*
dashboard.

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 achieve the creation of a monitoring dashboard that provides information about sentiment-emotion analyzed tweets. The panel has several widgets that, in spite of being independent with each other, implement interfaces which allows interaction between them. Besides, it is not limited to these widgets, instead there are some defined interfaces easily applied that let introduce new interactive elements.

All development has been based on GSI UPM Sefarad standard, which establishes the ecosystem defined in 2.4.3. Extending Sefarad's developments has been one of this thesis purposes, along with offering a good experience browsing analyzed tweet data. Furthermore, all elements that the dashboard implements can be used in other new developments based on Sefarad.

6.2 Achieved goals

- **Developing a map Polymer element with sentiment and emotion based layers**. There has been achieve a main data representation for the dashboard. With it it is possible to easily recognize the analyzed information and where is it located.
- **Developing new Sefarad widgets**. One important goal has been creating Polymer elements that allows to construct a dashboard complying the requirements defined in 3. Specifically, it has been developed two: a tweet list and a date slider bar.
  - **Tweet List Element** This element is basic to browse between all tweets displayed, and it allows a easy and intuitive interaction with the user when selecting items.
  - **Date Slider Element**. Since it is not desirable to change the deployment configuration each time a user want to get less data, a element for filtering was necessary. Finally, was has been achieved is a bar that allows movement and resizing for selecting a date range that will be added into dashboard's database query.
- **Developing a dashboard that integrates all previous elements**. It is necessary a union point between all widgets in order to get the data and pass it to them. This dashboard has implemented all interfaces defined by each one of the Polymer elements.
- **Luigi data pipeline**. The creation of pipeline to store and process geo-located tweets was a requisite in this project. This has been accomplished using Luigi and implementing all steps described in 4.2.1. Luigi's batch jobs are the entry point for using all elements that concerns the dashboard.

6.3 Problems faced

As in any other software project, the development faced several problems. Identifying these issues and producing a technically sound solution has been a challenge. Therefore, they are detailed here:

- **Technology crossed implementation**. A long list of technologies has been used in this project. It has been a challenge to face all of them one by one and, then, make them work together. The most problematic integration has been Leaflet within a Polymer element, since the pairing between Leaflet customization and Polymer properties is not trivial. Furthermore, some libraries have been modified with community pull requests code, because they are not not maintained anymore and they had problems working with other technologies.

- **Implementation of technologies in development**. Mainly, Sefarad and Polymer have advanced their development and suffered drastic API and conceptual changes during the development of this project. These changes implied refactoring big parts of the code code and integrating new technologies. On the other hand, although Polymer 2.0 version has been released and it includes many improvements, this project could not take advantage of it because some of the dependencies had not migrated to the new version yet.

- **Low dataset data quality**. Tweets can be posted from lot of different devices, by diverse kinds of people. Hence, they are very heterogeneous. For instance, they do are not always geo localted, and many of them are simply spam. Besides, after receiving Senpy analysis response of many tweets, it has been noticeable that approximately 85% of them are tagged with neutral emotion. For the sake of simplicity, all neutral tweets have been discarded. Consequently, the dataset needed was much bigger than expected.

6.4 Future work

There are several lines that can be followed to extend some of the this project's features. For the sake of brevity, we will mention only a highlight:

- **Improving filtering**. Tweet filtering options could be extended with more parameters. This could be done with interactive polymer elements that could select, for example, emotion tags or dataset's most influenciable Twitter usernames.

- **Twitter login**. With a personal Twitter login, the user could chose to be redirected to some tweet in Twitter web in order to follow the user ot browse responses.

- **Statistics widget**. An interesting option would be a new widget that dinamically shows dataset statistics as number of tweets, number of joy tagged tweets, most retweeted tweet. This could be implemented by integrating or extending an existing widget such as *number-chart*.

APPENDIX A

Component Integration and Demo Deployment

A.1 Modular Integration Among Polymer Components

So far, a total of three dashboard widgets have been described functional and logically. As it was mentioned before, one of this project development core purposes is the full reusability of each Polymer element. Lastly, this has been accomplished only introducing some dependencies that concern Polymer properties, any further internal logic will need changes.

Therefore, since every component can be understood as an independent library, each one is saved in an individual Git repository and registered separately in bower. Any can be installed from anywhere easily with bower:

```
$ bower install happymap
$ bower install date-slider
$ bower install tweet-list
```

In order to integrate HappyMap, Date Slider, or Tweet List in other project, it is recommended to read the provided documentation in Git repository's readme at *https://lab.cluster.gsi.dit.upm.es/*

dashboard.

A.2 Demo deployment

In order to let users try this project's dashboard in their own machine, a demo have been developed and a Docker deployment configuration have been provided. The demo is a simple *index.html* that imports and instantiates the dashboard Polymer component, which actually shows the content in the web page. For the deployment, it is necessary to have a *Dockerfile* and *docker-compose.yml* in the project root.

A.3 Dockerfile

This file shows Docker how to build the environment in which all project is going to be deployed.

```
FROM node:7.10.0
ENV NODE_PATH=/tmp/node_modules APP_NAME=TFG-JorgeGarciaCastano
RUN npm install -g http-server bower
ADD bower.json /usr/src/bower.json
RUN cd /usr/src && bower install --allow-root
ADD . /usr/src/app
WORKDIR /usr/src/app/
CMD [/usr/src/app/init.sh]
```

This lines meaning are, respectively:

1. The FROM instruction sets the Base Image for subsequent instructions. In this case, a Node.js image is chosen as server core.

2. The ENV instruction sets the environment variable ¡key¿ to the value ¡value¿. In this case, node.js path and app name are set.

3. The RUN instruction will execute any commands in a new layer on top of the current image and commit the results. In this case, Http-Server and Bower packages are installed via npm.

4. The ADD instruction copies new files, directories or remote file URLs from ¡src¿ and adds them to the filesystem of the image at the path ¡dest¿. In this case, bower.json file is added to images's filesystem in /usr/src/.

5. Bower dependencies are installed in /usr/src/ path.
6. All repository content is copied into /usr/src/app image's path.
7. The WORKDIR instruction sets the working directory for any other instructions that follow it in the Dockerfile. /usr/src/app/ path is set as working directory
8. The CMD instruction allows the execution of a command line and its main purpose is to provide defaults for an executing container. In this case, /usr/src/app/init.sh script is executed. init.sh is a shell script that copies all bower dependencies inside the app folder and launches the previously downloaded Node's Http-Server.

Once the *Dockerfile* is present in the project, there is only lack for one more file: *docker-compose.yml*.

A.3.1 docker-compose.yml

This file is the one that docker-compose will take as scheme for its service deployment.

```
version: '2'
services:
  sefarad:
    build: .
    ports:
     - 8080:8080
    volumes:
     - .:/usr/src/app
    networks:
      - sefarad-network
    depends_on:
      - elasticsearch
  elasticsearch:
    image: elasticsearch
    ports:
      - 9200:9200
      - 9300:9300
    volumes:
      - ./elasticsearch/nodes:/usr/share/elasticsearch/data/nodes
      - ./elasticsearch/config:/usr/share/elasticsearch/config
    networks:
      - sefarad-network
  luigi:
    build:
      context: luigi/
    volumes:
      - ./luigi:/usr/src/app
    networks:
      - sefarad-network
    depends_on:
      - elasticsearch
    environment:
      - PYTHONUNBUFFERED=0
networks:
  sefarad-network:
    driver: bridge
```

All services defined are detailed in the next points:

- Sefarad. This is actually the container that serves the whole application. It is mounted in /usr/src/app image's path and it exposes 8080 port as entrypoint for the dashboard.
- ElasticSearch. The latest ElasticSearch version is the base image for this container. It exposes 9200 and 9300 ports for database communication with Sefarad's container, and a couple of mounting points are defined for configuration and nodes.
- Luigi. Mounted in usr/src/app/luigi folder and built from luigi's repository folder. PYTHONUNBUFFERED environment variable is set to 0 in order to have the stdout of Luigi's pipeline at the terminal when calling docker-compose.

Sefarad, ElasticSearch, and Luigi are contained in the same Docker network, called sefarad-network, which allows a network communication between the three containers.

Bibliography

[1] Guillaume Gautreau. Jqrangeslider documentation http://ghusse.github.io/jqrangeslider/. 2017.

[2] Docker Inc. Docker-compose documentation https://docs.docker.com/compose/. 2017.

[3] Docker Inc. Docker documentation https://docs.docker.com. 2017.

[4] Docker Inc. Dockerhub repositories https://hub.docker.com. 2017.

[5] Object Management Group Inc. Uml documentation http://www.uml.org. 2017.

[6] Twitter Inc. Twitter api documentation https://dev.twitter.com/rest/public. 2017.

[7] Chrome organization. Polymer documentation https://www.polymer-project.org. 2017.

[8] WebComponents Organization. Webcomponents repositories https://www.webcomponents.org. 2017.

[9] Matt McMahon (The MITRE Corporation) Peter Dodds, Chris Danforth (Computational Story Lab). Andy Reagan (University of Vermont Complex Systems Center). Brian Tivnan. Hedonometer http://hedonometer.org. 2013.

[10] Juan Fernando Sánchez Rada. Sefarad documentation http://sefarad.readthedocs.io. 2017.

[11] Spotify. Luigi documentation http://luigi.readthedocs.io. 2017.

[12] J. Fernando Sánchez-Rada, Carlos A. Iglesias, Ignacio Corcuera-Platas, and Oscar Araque. Senpy: A pragmatic linked sentiment analysis framework. In *Proceedings DSAA 2016 Special Track on Emotion and Sentiment in Intelligent Systems and Big Social Data Analysis (SentISData)*. 00001.

[13] Alex MacCaw (Twitter). Bower documentation https://bower.io. 2017.

[14] Dr. Monica Stephens (Humboldt State University). Hatemap http://users.humboldt.edu/mstephens/hate/hate_map.html. 2017.

[15] Kuperman V. & Brysbaert M. Behav Res 45: 1191 Warriner, A.B. Norms of valence, arousal, and dominance for 13,915 english lemmas. 2013.