UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESIGN AND DEVELOPMENT OF A LYRICS EMOTION ANALYSIS SYSTEM FOR CREATIVE INDUSTRIES

JOSÉ MARÍA IZQUIERDO MORA

ENERO 2018

TRABAJO FIN DE GRADO

Título:	Diseño y desarrollo de un sistema para el análisis de los sentimientos de las líricas de las canciones para la industria creativa
Título (inglés):	Design and Development of a Lyrics Emotion Analysis System for Creative Industries
Autor:	José María Izquierdo Mora
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

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 FIN DE GRADO

DESIGN AND DEVELOPMENT OF A LYRICS EMOTION ANALYSIS SYSTEM FOR CREATIVE INDUSTRIES

José María Izquierdo Mora

Enero de 2018

Resumen

Como resultado del trabajo realizado a lo largo del proyecto cuyo objetivo ha sido diseñar y desarrollar una interfaz para el análisis de los sentimientos y emociones que producen las letras de las canciones.

La forma de conseguirlo, ha sido desarrolando un entorno de visualización para presentar los datos obtenidos de las letras de las canciones y analizarlos con Senpy. Este entorno de visualización se ha desarrollado con Polymer Web Components y D3.js.

El flujo de los datos ha sido entre musixmatch, Senpy y ElasticSearch. Se ha usado Luigi en este proceso ya que ayuda a construir flujos complejos de tareas agrupadas, Luigi se ha encagado de realizar el análisis de las letras y almacenarlos en ElasticSearch.

A continuación, se ha empleado D3.js para crear widgets interactivos que hacen que los datos sean fácilmente accesibles, estos widgets permitirán al usuario interactuar con ellos y filtrar los datos que le sean más interesantes. Para hacer la interfaz acorde al Material de la librería de Polymer Web Components.

Como resolución del proyecto, podemos realizar un amplio análisis de las letras, dependiendo de las canciones y artistas y de esta forma conocer las emociones y sentimientos que generan.

Palabras clave: Senpy, Letras, Sentimientos, Emociones, ElasticSearch, Sefarad, Análisis

Abstract

The way to achieve this, it has been developing a visualization environment to present the data obtained from the lyrics of the songs , and analyze them with Senpy. This visualization environment has been developed with Polymer Web Components and D3.js.

The data flow has been between musixmatch, Senpy and ElasticSearch. Luigi has been used in this process as it helps to build complex flows of clustered tasks, Luigi has gone from analyzing the lyrics to storing them in ElasticSearch.

Next, D3.js has been used to create interactive widgets that make data easily accessible, these widgets will allow the user to interact with them and filter the data that is most interesting to them. To make the interface according to the Material Designing Google and displaying dynamic data in widgets, the Polymer Web Components library has been used.

As a resolution of the project, we can make a broad analysis of the lyrics, depending on the songs and artists and thus know the emotions and feelings they generate.

Keywords: Senpy, Lyrics, Sentiments, Emotions, ElasticSearch, D3.js, Analysis

Agradecimientos

Gracias a mi tutor Carlos, por todo el apoyo y ayuda recibida durante el desarrollo del proyecto.

A toda mi familia, especialmente a mis padres, abuelos y mi hermano los cuales han estado conmigo durante toda la carrera.

También a todas esas personas con las que he compartido esta carrera, de las cuales me llevo una bonita amistad.

Gracias a todos vosotros.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Context

In our daily life, music is an essential part, in which we all take refuge at some point. This has been present in humans since we learned the language, has been part of numerous movements and always its lyrics have been accompanied by such movements and expressions.

As such, we can consider that we express the feelings through the music, since it will not be the same if we are sad, happy, etc. In this way, it would be viable to realize an analysis of the lyrics of the songs.

Emotion analysis can become a useful tool for creative industries to search, recommend, or assist in in composition processes.

The main objective of this project is to develop and deploy a dashboard that reflects the results of the analysis of the reactions and feelings of the desired songs. Transforming them into a simple way to understand and visualize for the user.

For the realization of the project, we will divide it into different phases, each with its corresponding tools. The first phase consists of a data scraper in the chosen song, trying to compile the lyrics of it, through musixmatch, to analyze it later. The result of this will be

stored in a json file, with a predefined structure. Then, this information will be collected in ElasticSearch and analyzed with the help of Senpy. Luigi, from Python, will help to make these questions. Finally, the results of the analysis will be displayed in a Dashboard managed by Sefarad, which will generate the visualization for the user. The programming languages of the project to use will be Python, HTML, CSS or JavaScript.

In summary, this project will allow the user to perform a complete analysis of the sentiment and the emotion that generate the songs, being able to choose the same ones.

1.2 Project goals

Ultimately, the project tries to show the future users the result of the analysis of the feelings that the lyrics of the songs produce. In this way, the project collects data and generates a pipeline to analyze feelings and emotions. Finally, everything we do, through interactive widgets for results.

The main objectives of the project are:

- Get the lyrics of the desired songs.
- Create a pipeline for analysis of emotions and feelings.
- Be able to display results in a dashboard made up of widgets.

1.3 Structure of this document

The structure of the chapters which will be described in the document will be briefly summarized below:
Chapter 1 puts in context the project developed. Also, briefly review the main objectives to be achieved.
Chapter 2 provides detailed information on the technologies used to achieve this project.
Chapter 3 details the design and implementation phases of the project. Furthermore, explains the components that make up the project. In addition to how the visualization module is designed. All the architecture.
Chapter 4 explains the details of the realization of our project, specifically the case study where the results will be analyzed.
Chapter 5 debates the problems encountered, the conclusions drawn from the project and

suggestions for future work.

CHAPTER 2

Enabling Technologies

2.1 Introduction

During this chapter, we provide detailed information on the technologies used to achieve this project.

To carry out the project, we have used the technologies developed in the Grupo de Sistemas Inteligentes (GSI), such as Sefarad or Senpy. The first, for the visualization of the analyzed data. The other, we need it for the analysis of feelings and emotions.

The way to extract the data has been made thanks to the Musixmatch API, once obtained the data that we have introduced in ElasticSearch through Luigi. Subsequently, these data will be the ones we will visualize.

2.2 Sefarad

Sefarad [4] an application designed to visualize data from SPARQL queries, directly at the end without needing to write more code. The way of visualizing such data, is done through widgets based on Polymer Web Components. It also offers the ability to create cores for

large collections of data.

Sefarad environment is divided into three dock containers, each focusing on one task, as we can see in the figure 2.1:

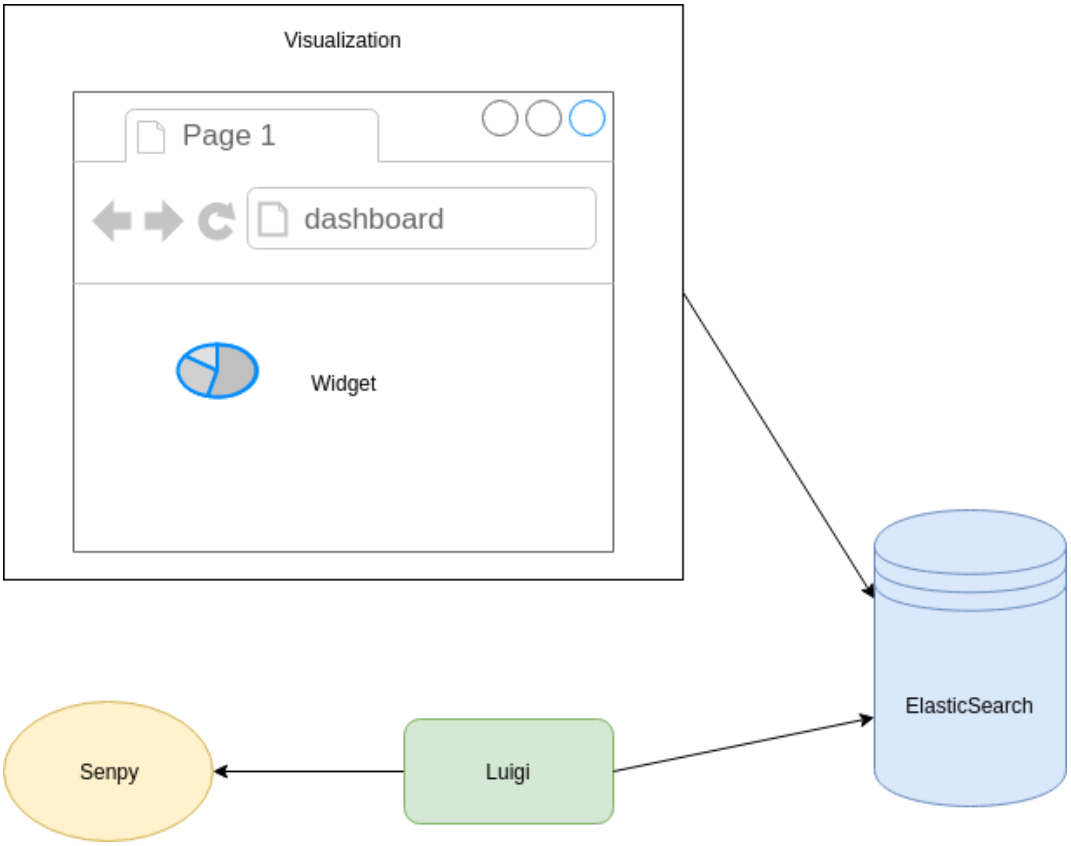

Figure 2.1: Sefarad architecture

2.3 Visualization

The purpose of this module is to represent the data being processed and to draw different graphs. This display is structured in different control panels. In addition, this dashboard is divided into different widgets. For this purpose we will use Polymer.

2.3.1 Polymer Web Components

Polymer [9] is a lightweight library built on top of web standards-based APIs for web components. Facilitates the creation of your customized html elements to your liking in a

more efficient and easy way by building more complex web applications.

Because it is based on the Web Components API built into the browser, Polymer elements are interoperable at the browser level and can be used with other frameworks or libraries that work with modern browsers.

With this technology we can take advantage of the elements created, can be particularly useful for building reusable user interface components. Instead of continually rebuilding an element in different frames and for different projects, it can be defined once and reused throughout its Project or any future project.

Polymer works with simple declarative syntax to create your own elements, using all standard web tools, such as element structure with HTML, CSS styles creation and interacting with JavaScript.

It is designed to be flexible and close to the web platform is based on the features of the web platform for creating custom elements.

In addition to the Polymer library, the project has a series of elemnts of predefined elements that you can drop onto a page, or use them as starting points for your own elements.This is classified in Polymer element catalog categories. The elements are listed below.

- ***Gold-elements.*** Elements built for e-commerce-specific use-cases, like checkout flows.
- ***Neon elements.*** This elemets are for transitions that draw attention and animations.
- ***Platinum-elements.*** Elements that take advantage of features to create a web page in a real web application, with push notifications and offline use.
- ***Molecules.*** This elements wraps other libraries in order to make them easier to use.
- ***App-elements.*** This elements are a set of components useful when building complete applications. They include components for functions such as routing, internationalization and data storage.
- ***Iron-Elements.*** This elements are basic building blocks for creating an application
- ***Paper-elements.*** User interface components designed to implement Google's material design orders.
- ***Google-web-components.*** This elements as the name implies a collection of web components for Google APIs and services

2.4 ElasticSearch

Elasticsearch [3] is an analytical RESTful search and distributed engine of free code under the apache licenses, being able to realize great amount of use cases. Everything from a centralized form, that is, from the core of the stack in order to discover what is expected and what is unexpected.

It gets answers instantly, which means that the relationship with the data changes. It can iterate and cover more ground.

It is implemented in inverted indexes with finite state transducers for full-text queries, BKD trees to store numerical and geo data, and a column store for analysis.

Being indexed can take advantage of and access all your data at great speeds.

Use Lucene and all its functions must be available through JSON and Java API. Admits face and percolocation, this can be beneficial if we want to know if new documents

2.4.1 Searching

Search queries are performed through the API and the results that match that search or query are retrieved.

As a parameter:

```
http://localhost:9200/music23/_search?pretty
```

As a request body:

```
"hits" : {
"total" : 4,
"max_score" : 1.0,
"hits" : [
{
"_index" : "music23",
"_type" : "prueba",
"_id" : "793365265",
"_score" : 1.0,
"_source" : {
"id" : 793365265,
"name" : "Michael Jackson",
"titulo" : "this is it",
"text" : "Ready?<br>One, two, three, ahh!<br>This is it, here I stand<br>I'
    m the light of the world<br>I feel grand<br>Got this love, I can feel<
```

```
    br>And I know, yes for sure<br>It is real<br><br>And it feels as though
     I've seen your face a thousand times<br>And you said you really know
    me too yourself<br>And I know that you have got addicted with their
    eyes<br>But you say you're gonna leave it for yourself<br>Ohhh!<br>I
    never heard a single word about you<br>Falling in love wasn't my plan<
    br>I never thought that I would be your lover<br><br>Come on baby just
    understand<br><br>This is it<br>I can say<br>...<br><br>",
"sentiment" : "marl:Positive",
"polarity" : "1"
}
},
```


2.5 Luigi

Luigi [10] is a Python package that helps you build complex work pipes. It manages workflow management, visualization, fault handling, command line integration and it manages much more.

Luigi is similar to GNU Make, this one has some tasks and in turn they can have dependencies on other tasks. There are also some similarities with Oozie and Azkaban. The big difference is that Luigi is not only designed for Hadoop, and can easily be extended with other types of tasks.

Luigi is in Python. Instead of the XML configuration or similar external data files, the dependency graph is stati fi ed within Python. This facilitates the creation of complex graphs of task dependencies. However, the workflow can trigger things that are not in Python, such as running Pig scripts.

As an example, this tool internally in Spotify is used to execute thousands of tasks every day, organized in complex dependency graphs. are usually Hadoop works, those made in these tasks. It provides an infrastructure that powers all sorts of things, such as recommendations, classification lists, and test analysis.

2.6 Senpy

Senpy [6] is a framework for sentiment and emotion analysis services. Services created with senpy are interchangeable and easy to use because they share the API. It is based on NIF,

Marl and Onyx [2] vocabularies.Furthermore, it simplifies the development of the service.

Senpy implements all common blocks, so that developers can focus on what really matters: large analysis algorithms that solve real problems.

Senpy is a framework that converts your algorithm of feeling or emotion analysis into a complete semantic service. Senpy deals with:

- ***Interface.*** Parameter validation, error handling.
- ***Formatting.*** JSON-LD entry and exit, Turtle / n-triples, or simple text entry.
- ***Linked data.***Senile results are scored semantically, using a series of well-established vocabularies and default URIs.
- ***User interface.*** A web user interface where people can explore their service and try different configurations.
- ***A client to interact with the service.*** Available in Python.

Finally, as far as architecture is concerned, the main component of a sentiment analysis service is the algorithm itself. However, for the algorithm to work, it needs to obtain the user's appropriate parameters, the results according to the API, interact with the user when errors occur or more information is needed.

2.7 Musixmatch

Musixmatxh[1] is a catalog of letters with more than 12.4 million letters in 50 languages. It is accessible on Windows and Mac through Spotify, as well as on mobile applications for operating systems such as iOS, Android and Windows Phone.

Musixmatch shows lyrics on the screen to see the music that is playing in time. In its native applications, it supports the ability to scan all songs in a user's music library and find lyrics for them, as well as to be used as a music player.

Its API-enabled allows website owners and mobile application developers to legally display and monetize the letters in their database.

For your use as a developer you will need an API key, a mandatory parameter for most API calls. It is a personal identifier and must be kept secret. Once the API key is obtained,

[1]https://developer.musixmatch.com/documentation

the developers can now exploit their methods and make all the calls they want as long as the lyrics of the songs are registered.

CHAPTER 3

Architecture

3.1 Introduction

In chapter three, we introduce the composition of the project architecture. A general part is explained with all the modules that form it. Then, we explain each module in detail separately. Finally, parts of what we have created are taught, such as the pipeline responsible for data extraction.

3.2 Architecture

In this section we explain the modules presented in the project architecture, as we can see in the figure shown below.

- ***Search System:*** This is one of the main parts of the project, where the data of the Musixmatch API[1] [5] songs are obteined.
- ***Orchestrator System:*** Through Luigi[10] we build a pipeline to connect the different modules, the search system with the analysis and indexing system.

[1] https://developer.musixmatch.com/documentation

- ***Analysis System:*** In our project, we use Senpy [6] to analyze the feelings and emotions of the songs obtained.
- ***Index System:***In this module, the data of the songs obtained in the other modules in Elasticsearch [**?**] are indexed.
- ***Visualisation System:*** This is another important part of the project, it reflects the results obtained, since it is responsible for processing and displaying the data.

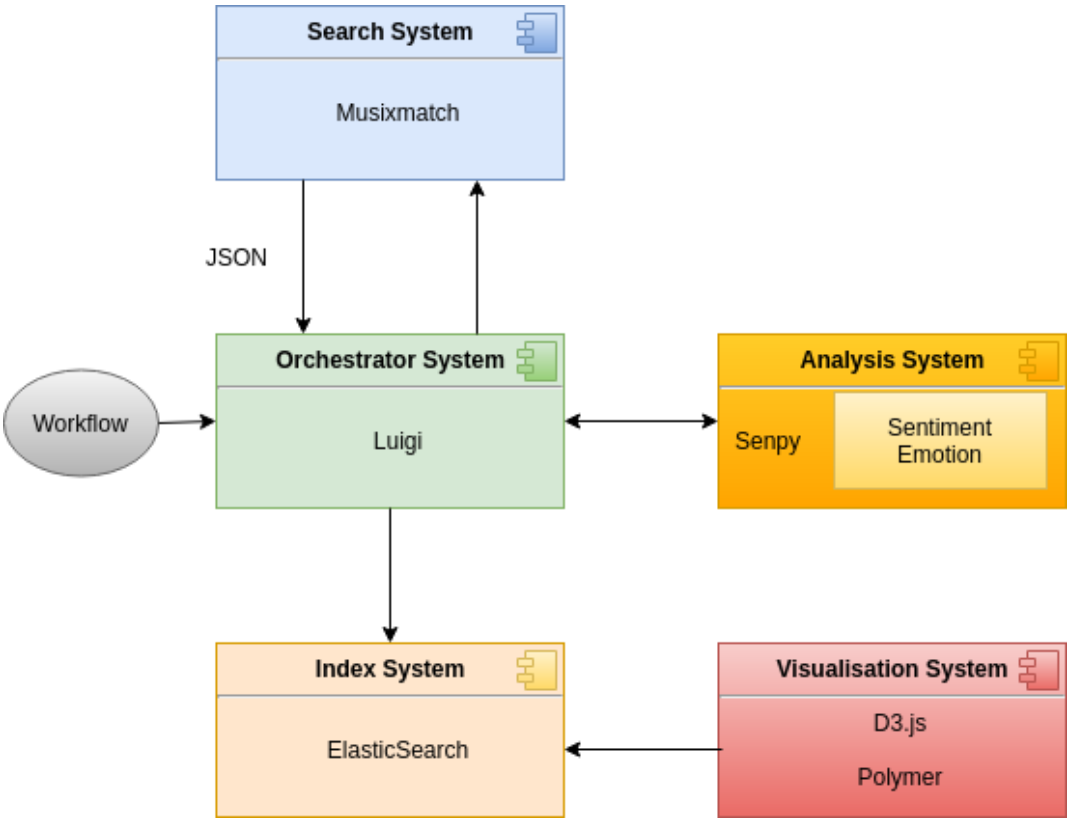

Figure 3.1: Architecture

Once the general modules have been described, the sub-modules of the architecture of our project are detailed in the following points.

3.3 Search System

This is one of the main parts of the project, as we said, because it is responsible for extracting the data we want to analyze.

The technology used to carry out this project, as explained in chapter 2 is Musixmatch, through the developed Python script that can be seen in the following code sample.

It is important to highlight that it is necessary to introduce a new parameter when using this script because the MusixMatch API requires the use of an "apiKey" to access its tools. This parameter is called in the code ***apikeyMusixmatch***. Another variable ***apiurlMusixmatch*** is also used, which runs the required url.

For our study of song lyrics, four artists of different styles and musical genres have been chosen to compare the data from different perspectives. The chosen artists have been Michael Jackson, Joaquin Sabina, Extremoduro and Julio Iglesias

Next, the method from which we obtain the data of the songs are shown. The way to process it is the following:

- We introduce in the method the name of the artist and the corresponding song.
- The method obtains the data from the Musixmatch API, and also transforms it into JSON format.
- Finally, we get the results in our JSON (songs.json).

```
import urllib.request, urllib.error, urllib.parse
import json
import socket
import random
apikeyMusixmatch =  'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
apiurlMusixmatch = 'http://api.musixmatch.com/ws/1.1/'

#This method returns the data of the songs in a json, they are passed as
    parameters the name of the song and the artist
def song_lyric(song_name,artist_name):
  while True:
  #Make a request to the API with the query by calling it with the
      parameters mentioned above and the lyrics.get method
  querystring = apiurlMusixmatch + "matcher.lyrics.get?q_track=" + urllib.
      parse.quote(song_name) + "&q_artist=" + urllib.parse.quote(
      artist_name) +"&apikey=" + apikeyMusixmatch + "&format=json&
      f_has_lyrics=1"
  request = urllib.request.Request(querystring)
  request.add_header("User-Agent", "curl/7.9.8 (i686-pc-linux-gnu) libcurl
      7.9.8 (OpenSSL 0.9.6b) (ipv6 enabled)")
  while True:
    #An API response timeout is established
```

```
  try:
  response = urllib.request.urlopen(request, timeout=4) #timeout
  raw = response.read()
  except socket.timeout:
  print("Timeout raised and caught")
  continue
  break
#The data is collected in JSON format and a random Id is established
json_obj = json.loads(raw.decode('utf-8'))
body = json_obj["message"]["body"]["lyrics"]["lyrics_body"]
copyright = json_obj["message"]["body"]["lyrics"]["lyrics_copyright"]
tracking_url = json_obj["message"]["body"]["lyrics"]["html_tracking_url"]
id_json = random.randrange(1, 1000000000,1 )
print(tracking_url)
lyrics_tracking(tracking_url)

#A JSON (songs.json) is generated with the desired data and the lyrics of
     the song are returned
from pprint import pprint
body2 = body.replace("\n","<br>")
body.split("...")
print(body)
data = json.dumps({"id": id_json,"name":artist_name, "titulo": song_name
    ,"text":body2},)
print(data)
with open('\home\jose\sefarad\luigi\songs.json', 'w') as outfile:
  outfile.write(data)
return (body)
```

Finally, once the JSON is obtained, the search system concludes, having obtained the necessary data to be able to proceed to the next step.

3.4 Orchestrator System

The chosen orchestrator system is Luigi [10], a Python module developed by Spotify that is responsible for the resolution of dependencies, the analais of the lyrics of the songs, etc.

Luigi makes this work through a script that describes the pipeline to follow. All this, is described in the workflow that is detailed in the next point.

3.4.1 Workflow

In the workflow, the tasks that form the pipeline are described, in our case this is collected in a script called buildPipeline.py that has been developed using the Luigi orchestator.

As you can see in figure (3.3) the phases that make up the process are shown. This phases are divided into FetchDataTask, SentimentTask, EmotionTask and Elasticsearch.

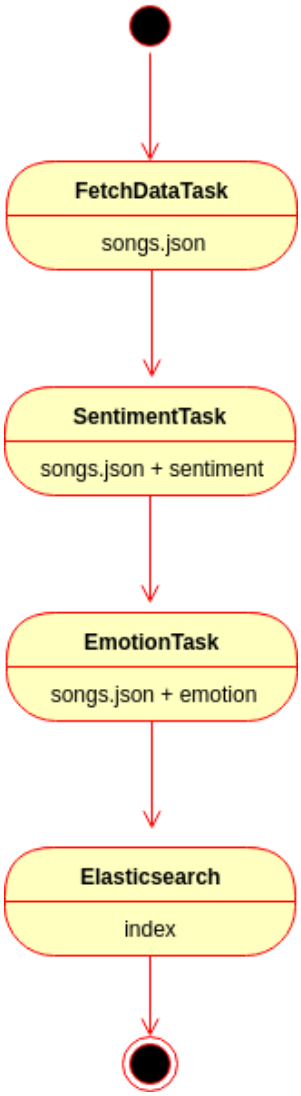

Figure 3.2: Pipeline Phases

By entering the following commands, we managed to execute our workflow.

Commands to execute the workflow

```
python3 -m luigi --module buildPipeline Elasticsearch -- index music --doc
    type results --filename songs.json --local scheduler
```

In this way, the orchestrator Luigi indexes music in the Elasticsearch database, after executing the buildPipeline.py script, performing the tasks that are in it using the songs.json file as a data source.

While executing Luigi's previous command, we made a couple of captures that can be seen in the list of tasks (3.3) and (3.4), where you can see the dependencies graphically and the tasks that are being executed.

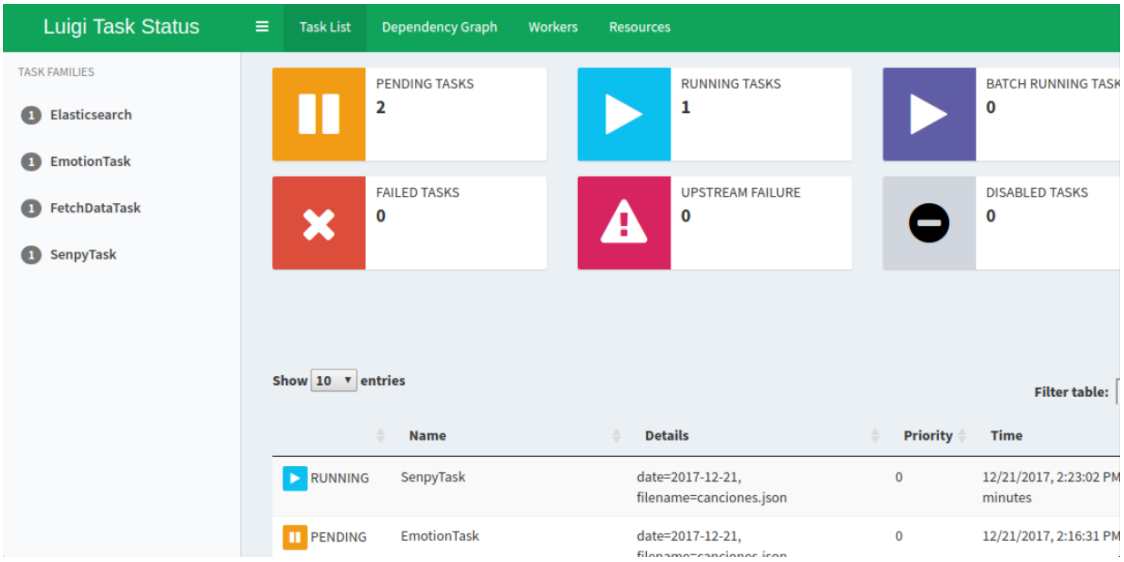

Figure 3.3: Pipeline

In the list of tasks (3.3), we can check the status of the pipeline, if any task has failed, which is running or which are pending, each task is updating status as it is running.

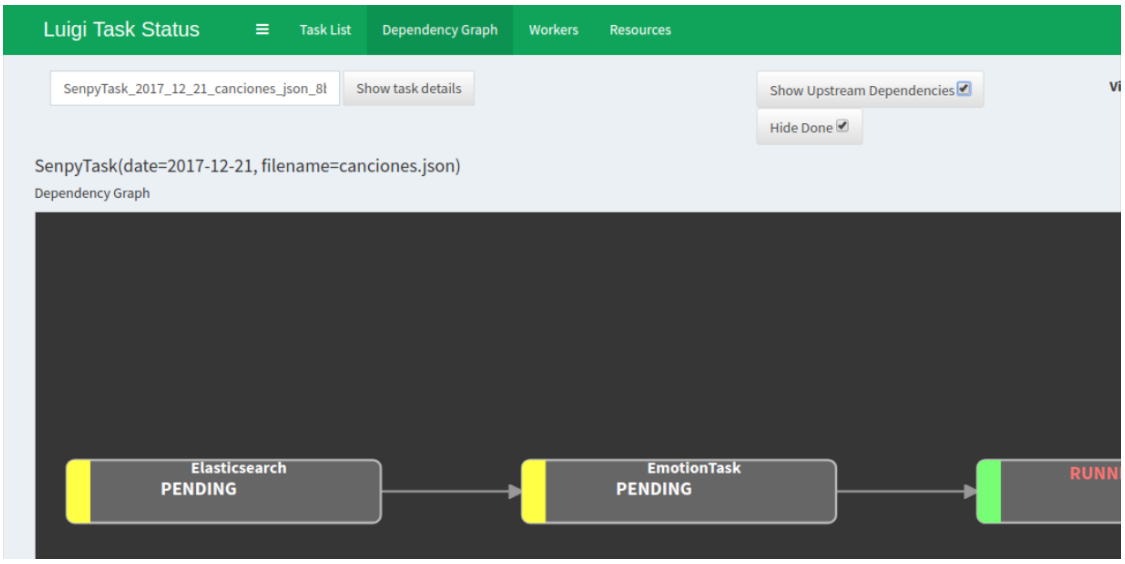

Figure 3.4: Dependency graph

In the dependency graph (3.4),we can observe a graph while performing the task of feelings, while waiting for the tasks corresponding to emotions and storage. Because until one is over, the other can not start.

3.4.1.1 FetchDataTask

FetchDataTask is the first task in the workflow whose main objective is to read the json file. As we can see below, the Python code responsible for this task is detailed.

```
class FetchDataTask(luigi.Task):
"""
Generates a local file containing  elements of data in JSON format.
"""

filename = luigi.Parameter()



def run(self):
"""
Writes data in JSON format into the task's output target.
The data objects have the following attributes:
* `_id` is the default Elasticsearch id field,
* `text`: the text,

"""
file = self.filename

with open(file) as f:
j = json.load(f)
for i in j:
i["_id"] = i["id"]
with self.output().open('w') as output:
json.dump(j, output)
output.write('\n')



def output(self):
"""
Returns the target output for this task.
In this case, a successful execution of this task will create a file on the
     local filesystem.
:return: the target output for this task.
:rtype: object (:py:class:`luigi.target.Target`)
"""
return luigi.LocalTarget(path='/tmp/_docs-%s.json' % self.filename)
```

3.4.1.2 SentimentTask

This task loads the data from the previous task and sends it to Senpy to analyze the recovered data. In this process, the feelings expressed by the lyrics of the songs that make up our project are analyzed. Once analyzed, a parameter is introduced in the data with the result of the analysis.

We can observe the code responsible for carrying out this task.

```
class SentimentTask(luigi.Task):
"""
This task loads data fetched with previous task and send it to Senpy tool
    in order to analyze
data retrieved and check sentiments expressed.
"""

#date = luigi.Parameter()
filename= luigi.Parameter()
#file = str(random.randint(0,10000)) + datetime.datetime.now().strftime("%Y
    -%m-%d-%H-%M-%S")



def requires(self):
"""
This task's dependencies:
* :py:class:`~.FetchDataTask`
:return: object (:py:class:`luigi.task.Task`)
"""
return FetchDataTask(self.filename)

def output(self):
"""
Returns the target output for this task.
In this case, a successful execution of this task will create a file on the
     local filesystem.
:return: the target output for this task.
:rtype: object (:py:class:`luigi.target.Target`)
"""
return luigi.LocalTarget(path='/tmp/analyzed-%s.jsonld' % self.filename)



def run(self):
"""
Send data to Senpy tool and retrieve it analyzed. Store data in a json file
    .
```

```
"""
with self.output().open('w') as output:
with self.input().open('r') as infile:
j = json.load(infile)
for i in j:
r = requests.get('http://test.senpy.cluster.gsi.dit.upm.es/api/?algo=
    sentiment-tass&i=%s' % i["text"])
response = r.content.decode('utf-8')
response_json = json.loads(response)
i["_id"] = i["id"]
#i["analysis"] = response_json
i["sentiment"] = response_json["entries"][0]["sentiments"][0]["marl:
    hasPolarity"]
i["polarity"] = response_json["entries"][0]["sentiments"][0]["marl:
    polarityValue"]
output.write(json.dumps(i))
#print(i)
output.write('\n')
```

3.4.1.3 EmotionTask

In this task, once we have the data of the previous one, a process similar to the previous one takes place. The data is sent to the Senpy tool, where the lyrics of the songs are analyzed by their emotions.

Once this process is finished, they are stored in a temporary file that is inserted in the next task.

3.4.1.4 Elasticsearch

Elasticsearch is the last task of the workflow. Load the contents of the file into an Elasticsearch index. In this case, it indexes all the data analyzed in music.

Now we can keep the code corresponding to that task.

```
class Elasticsearch(CopyToIndex):
"""
This task loads JSON data contained in a :py:class:`luigi.target.Target`
    into an ElasticSearch index.
This task's input will the target returned by :py:meth:`~.Senpy.output`.
This class uses :py:meth:`luigi.contrib.esindex.CopyToIndex.run`.
"""
date = luigi.DateParameter(default=datetime.date.today())
```

```
filename = luigi.Parameter()

#: the name of the index in ElasticSearch to be updated.
index = luigi.Parameter()
#: the name of the document type.
doc_type = luigi.Parameter()
#: the host running the ElasticSearch service.
host = 'localhost'
#: the port used by the ElasticSearch service.
port = 9200

def requires(self):
"""
This task's dependencies:
* :py:class:`~.SenpyTask`
:return: object (:py:class:`luigi.task.Task`)
"""
return EmotionTask(self.date,self.filename)
```

3.5 Analysis System

Senpy is responsible for the analysis of feelings and emotions in Python. This server has been developed by the Intelligent Systems Group (GSI), a group belonging to the Polytechnic University of Madrid. Specifically the plug in used have been created by Ignacio Morcuera[6]:

- ***Sentiment-Tass:*** This plug-in is used to perform the sentiment analysis. A distinction is made between positive and negative feelings.
- ***Emotion-ANEW:*** This other plugin is used to perform the analysis of emotions. Available emotions are anger, disgust, negative-fear, joy, neutral-emotion and sadness.

3.6 Index System

This system indexes all the lyrics of songs obtained and analyzed in the previous tasks. It is composed of Elasticsearch, the search server that connects the lyrics of the songs and their data with the visualization system.

In our case, we have chosen the easiest and fastest way to add the data to the Elaticsearch index. The form to do it has been as already explained in the previous point through a task of Luigi, where you can also see how to do it.

In the same way we choose to be more effective, to make a single index for the four artists of our project, instead of four, all grouped in the music index.

Finally, in Elasticsearch you can consult the index state of our database through the following URL:

```
http://localhost:9200/_cat/indices
```

We can see the results obtained with the previous URL, in the following figure(3.5).

```
yellow open music      U-d8fArcR1uNuOtfZpMnZQ 5 1  136 0 278.1kb 278.1kb
yellow open update_log VMNV86KfSSq6_9sjG_4RHg 5 1   23 0 124.1kb 124.1kb
```

Figure 3.5: Index

3.7 Visualisation System

The last model of our project, but one of the most important parts due to being the part that the end user visualizes and understands of our project, without which all the above would make sense [8].

This visualisation system is based on Polymer Web Components[2] and is developed in Sefarad 4.0. The Polymer library [9] is based on the design of Google materials and supports the creation of new components or widgets. In addition, we have used the D3 library. js [1] to design new web components.

3.7.1 Mock up

For the realization of the visual part of our project, we previously needed to create a sketch or mock up, which guides us to make our dashboard. It is important to carry out this previous task, since although modifications could be made to possible changes of opinion, it is very useful to be able to start the visualization system.

For our project, we wanted to show the feelings and emotions produced by the lyrics of the songs on a website, interactively and as comprehensively as possible for the user.

Analyzing the possible solutions, we decided to use different graphics that allow the user to compare emotions and feelings between different artists, songs, etc.

We also realized the need to show structured information, so we choose a taskbar with easy access, to be able to scroll through the different views of the application. The reason is to make a logical scheme for the website.

After making this previous analysis about what we wanted on our website, we used the draw.io[4] tool to make two mock ups, which shows us the main ideas we want on our website.

[2]https://www.polymer-project.org/1.0/
[4]http://draw.io

In the figure (3.6) we can see a mock up of the first tab, where a search engine is displayed on the main songs of our board.

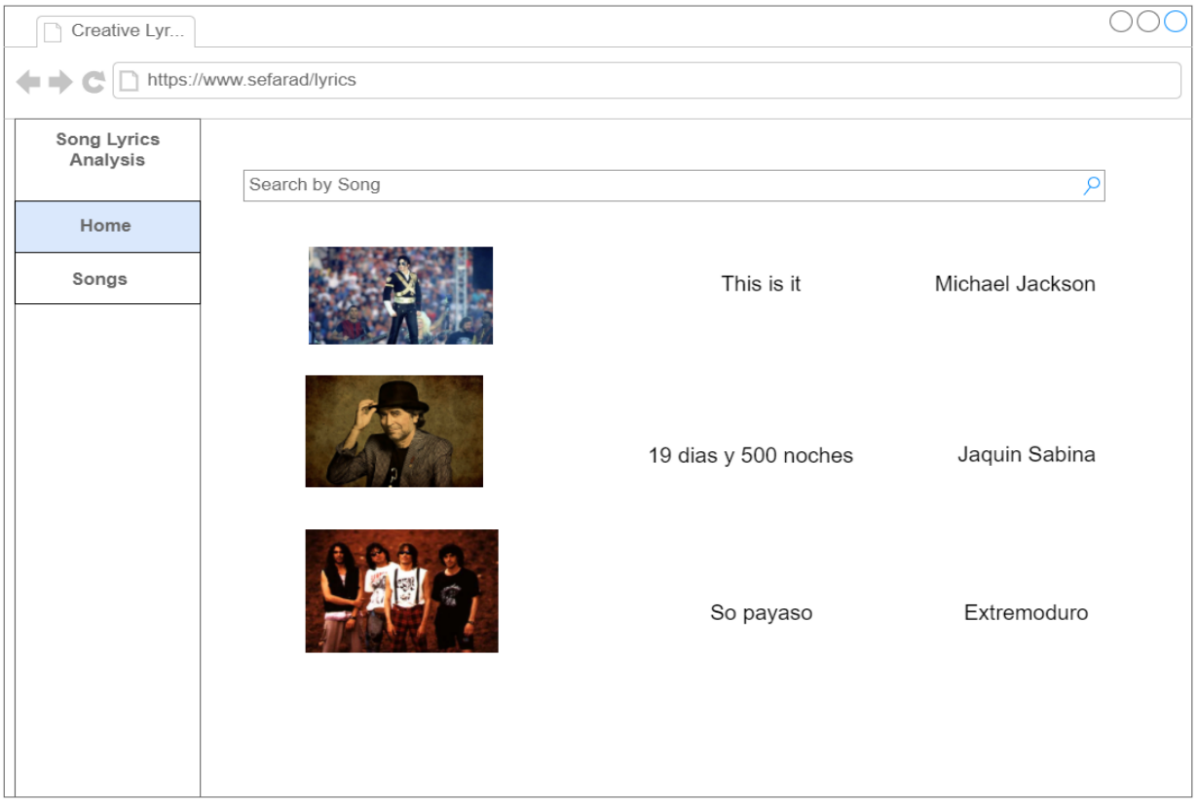

Figure 3.6: Home tab mock-up

And in the second figure (3.7), all the collected analyzes are shown, by feelings and emotions. You can also appreciate the separation that can be made by artists and songs.

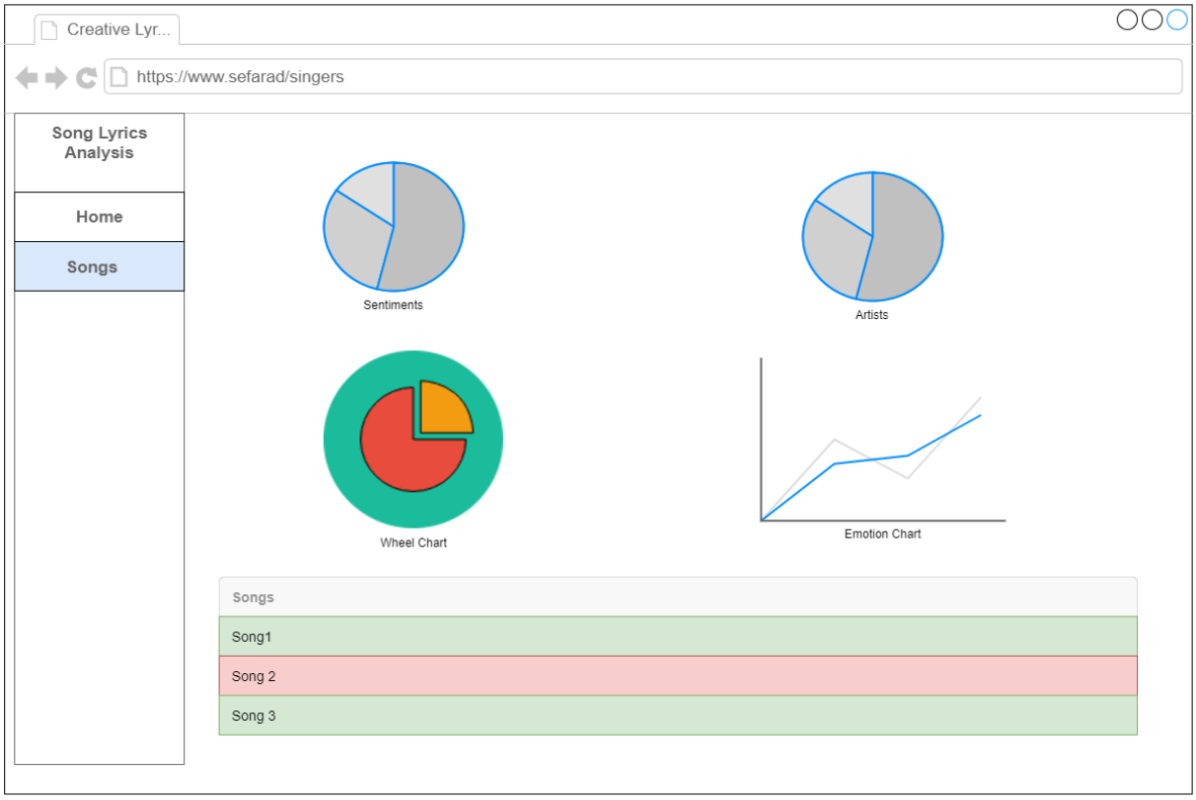

Figure 3.7: Songs tab mock-up

3.7.2 Widgets

The widgets provide information depending on the data that is introduced to the songs. These widgets are web components created using D3.js.

We had to adapt some widgets because the catalog of polymer elements was not enough to represent the graphics. Next, all created widgets are displayed.

3.7.2.1 Google Sentiment Chart

This widget is used to classify songs by feelings, positive or negative. It is a very useful widget because mixed with others can be very useful, being able to compare songs along with other parameters. It admits a series of parameters:

- Query: The parameter used the search box automatically.
- Title: Title: The title is the same as shown in the widget bar.
- Field: It passes the field for which he wants to filter, in this case sentiment.
- Data: the data is entered as a parameter.

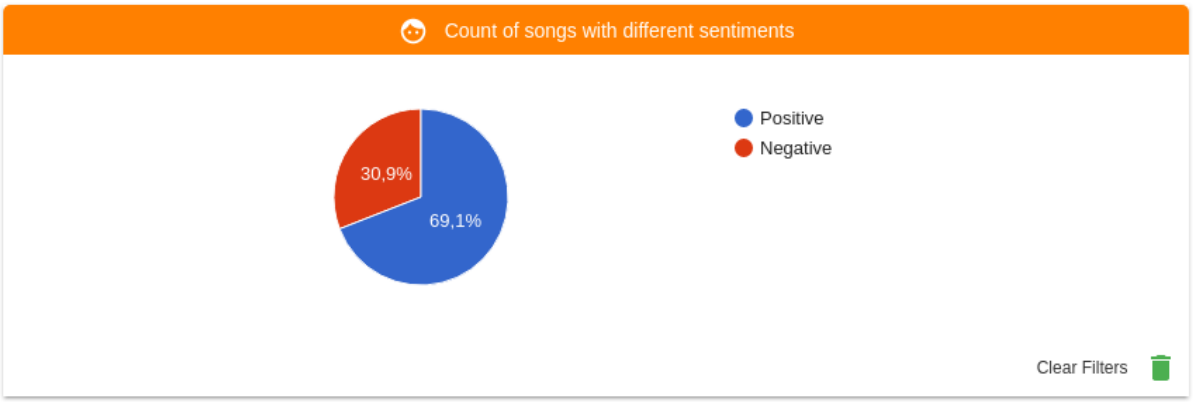

Figure 3.8: Google Sentiment Chart

3.7.2.2 Google Singers Chart

The widget described below is very similar to the previous one (3.7.2.1). Classify the songs by artists instead of their sentiments. In this way, the parameters that the widget supports are the same, except field, where the artist is passed.

It is important to note, that this widget combined with others like the previous one, serves us to classify by its artists and at the same time by sentiments. The same with the rest of widgets.

3.7.3 Wheel Sentiment

This widget is formed by a wheel divided by three circles, the main one classified by artists, within those artists is classified in turn by positive and negative feelings with the colors green and red respectively. And to know what song is treated in its outermost layer are the lyrics of the songs, as shown in the figure (3.9). It admits a series of parameters:

- Query: The parameter used the search box automatically.
- Title: Title: The title is the same as shown in the widget bar.
- Field: It passes the field for which he wants to filter, in this case the lyrics of the songs.
- Data: the data is entered as a parameter.

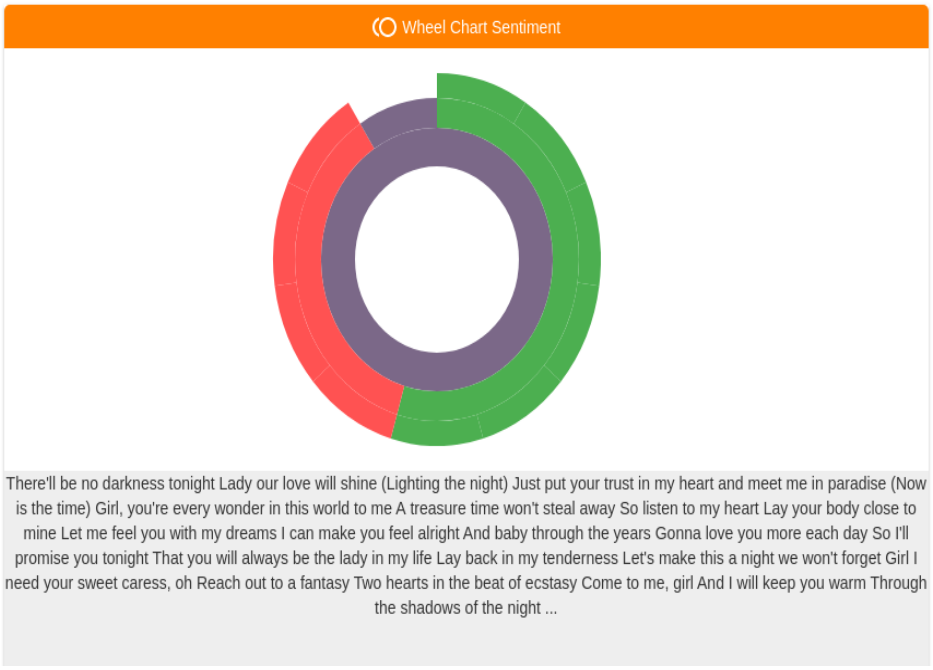

Figure 3.9: Wheel Sentiment

Like other widgets when filtered by other parameters in other wigets, it reflects those filters.

3.7.3.1 Spider Emtion

The following widget is used to classify the songs according to the emotions that represent them. In this case it is a table of axes, where in each axis the predominant emtion is shown. It admits a series of parameters:

- Query: The parameter used the search box automatically.
- Field: It passes the field for which he wants to filter, in this case the lyrics of the songs.
- Data: the data is entered as a parameter.

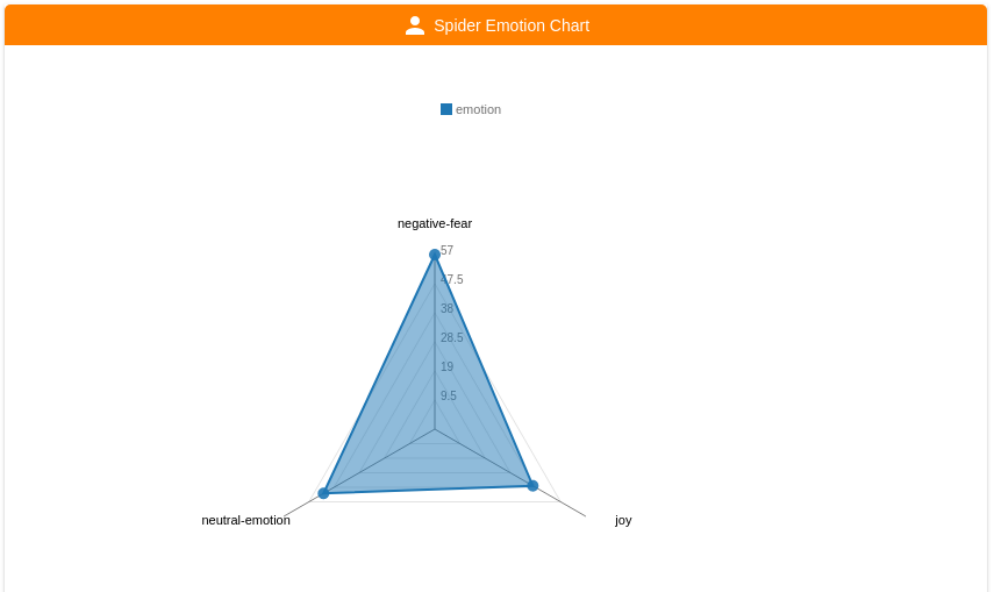

Figure 3.10: Spider Emotion

Like other widgets when filtered by other parameters in other wigets, it reflects those filters.

3.7.3.2 Songs Chart

This widget is one of the most representative, it shows the songs with as much data as possible. In them the songs are represented by their artist represented by a photograph and with the name of the same. In addition, the title of the song in particular is shown.

It is hidden the lyrics of the songs in the bottom of the container of each song, is shown by clicking on the widget. The color of each container varies according to the feeling, being green if it is positive and red if it is negative. It admits a series of parameters:

- Query: The parameter used the search box automatically.
- Title: Title: The title is the same as shown in the widget bar.
- Data: the data is entered as a parameter.

Figure 3.11: Songs Chart

Like other widgets when filtered by other parameters in other wigets, it reflects those filters.

3.7.3.3 Song Searcher

This widget is used to filter the data by songs, that is, it is a song search engine in our data. It admits a series of parameters:

- Query: The parameter used the search box automatically.
- Title: Title: The title is the same as shown in the widget bar.

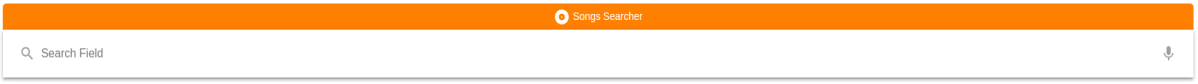

Figure 3.12: Song Searcher

When doing a search, make a query to the Elasticsearch database. The received data is filtered and only the selected song is displayed.

3.7.4 Dashboard Tabs

The visualization system uses tabs to separate the tabs, since in each tab different types of data are shown.

To make the tabs we have used a polymer element, specifically paper tabs, we describe its function and each of the tabs and their contents.

3.7.4.1 Papers tab

This element organizes the menu in which they can appear or the main tab of our web, or a secondary tab where all the analysis of the web is collected in depth.

The following figure (3.13) shows the result, where we can see the tab Home tab or Songs tab.

Figure 3.13: Papers tab

3.7.4.2 Home tab

The home tab has been designed to show the user a main window where to see the songs of the applications.

It has been considered that for the main window, it is easier to use with the following components:

- Songs Searcher: We use the search engine to find the song that we want to show, since the search filter is filtered by the title of the song.
- Songs Chart: Which is used to show the data of the songs, in this way we obtain the letter a photo of the artist, the title and the name of the artist.

3.7.5 Songs tab

This second tab has been designed to show the user a more detailed analysis of the analysis that the song has had.

In this way we have considered it more convenient to include the following components in this tab:

- Google Sentiment Chart: In this widget the percentages of the feelings of the parameters that we have selected at that moment are shown.
- Google Singers Chart: In this widget the percentages of the singers of the parameters that we have selected at that moment are shown.
- Wheel Chart: This widget comparatively shows different types of artists classified according to their feelings.
- Spider Emotion Chart: In this widget comparatively different types of emotions that represent those songs are shown.
- Songs Chart: Which is used to show the data of the songs, in this way we obtain the letter a photo of the artist, the title and the name of the artist.

CHAPTER 4

Case study

4.1 Introduction

In this chapter, we describe the procedure that follows the extraction of data, which results reflects the analysis of data, the creation of the visualization system and the complete functioning of our system. Finally, the chapter close with a conclusion of the thesis.

The main actor in our case is the end user who uses our website with the aim of finding the information and analysis he wants about the lyrics of the songs.

4.2 Extracting data

To relate the extraction of data, as it was said in previous chapters, we have used the Musixmatch API to perform this task. Once the tool was chosen, we decided what data we wanted to extract. For this we decided what artists, songs and what period of time we wanted to measure.

Next, it shows that we finally chose for our project.

- **Artists:** It's about Michael Jackson, Joaquín Sabina, Julio Iglesias and Extremoduro. Different artists with different genres recognized internationally and nationally, which provide musical variety.
- **Songs:** The chosen songs are based on the following albums, chosen so that they belong to a similar time and adjusting to the same number of songs per author, to make a fair study among the four artists.
  - **Michael Jackson:** "Thriller" (1982), "Bad" (1987) and "Dangerous" (1991).
  - **Joaquín Sabina:** "Física y Química" (1992), "Dímelo en la calle" (2002) and "Dos pájaros de un tiro" (2007).
  - **Julio Iglesias:** "De niña a mujer" (1981), "Momentos" (1982) and "Tango" (1996).
  - **Extremoduro:** "Rock transgresivo" (1989), "Agila" (1996) and "Yo, minoría absoluta" (2002).
- **Time:** The period of time chosen varies between 1981 and 2007, although the majority of songs are between the decades of the 80s and the 90s.

4.3 Analyzing data

After the extraction of the data, as we said before, the turn now falls to the analysis of them. For this, we had two tasks in our pipeline that we passed to Senpy to perform an analysis of the letters by sentimients and emotions. The result has been:

- **Sentiments:** Positive (94) and negative (42).
- **Emotions:** Joy (37), neutral-emotion (42) and negative-fear (57).

4.4 Indexing data

The next step to the analysis is the storage of said data and its indexation. To do this, through another task in our pipeline, the data is indexed in Elasticsearch. We decided to use a single index for our data, since it did not make sense to enter several, being able to store all the data in a single one, where collecting that data is done easily and quickly.

In our case, for the "music" index, the necessary parameters have been introduced: id, title of the song, artist, lyrics, feeling, polarity and emotion.

4.5 Displaying data

For the visualization of our project, we use a dashboard formed by tabs and interactive widgets. Once the user accesses our website, he finds the Home tab.

In this figure (4.1) shown below, we see the Home tab, where we wanted to create the impression of a conventional website with a list of songs and a search engine to find each of them.

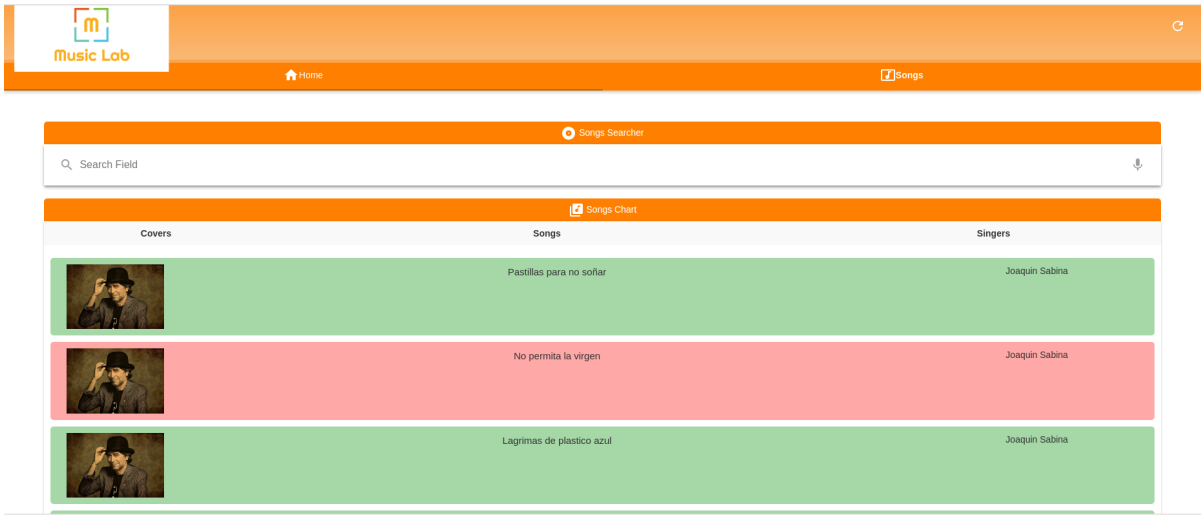

Figure 4.1: Home tab

The first widget, the song searcher, is very useful for the user, there are so many songs on our website, it is logical to introduce a search engine that filters the name of the song on the dashboard and shows us only the desired song. The default function is also useful, which restores the default values.

On the other hand, we consider important the need to show the lyrics of the songs with their specific title, artist and photo of it. For this we incorporate in the initial tab the widget created song chart, where the background of the space dedicated to the song is colored according to the feeling of that song.

In the figure (4.2), (4.3) and (4.4) the main tab of the application are shown, where we find most of the analysis. In it are introduced the main widgets that help us to observe results in different ways.

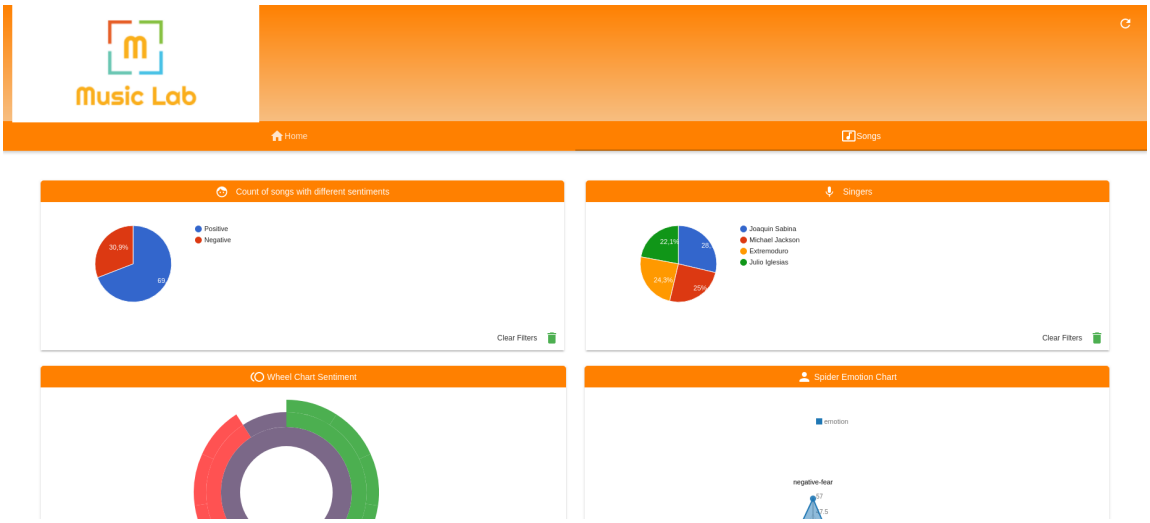

Figure 4.2: Songs tab part one

In relation to the songs, the need arises in our interactive website to be able to classify all our songs in positive and negative and in the different artists. For this reason, two widgets (Google Chart) have been created to classify by sentiment and by singers. The combination of both is very powerful because we can filter the songs by the artists and their feelings, which means that we can study each artist in depth.

In this way, we find that Michael Jackson has 61.8% of his positive letters, compared to 38.2% of negatives. For his part, Joaquín Sabina is 87.2% compared to 12, 8%. Julio Iglesias has a ratio of 63.3% to 36.7%. Finally, Extremoduro has 84.8% of positive letters compared to 15.2%.

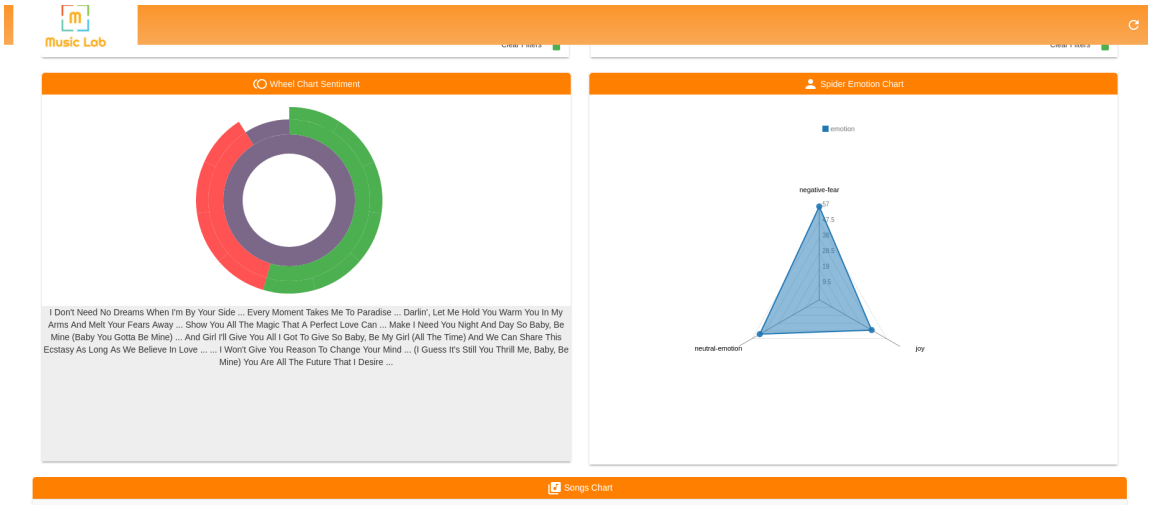

Figure 4.3: Songs tab part two

In this tab we also include the two widgets that let us compare feelings between the four artists and their emotions. On the one hand we have the wheel wheel widget that allows us to compare between different artists, since this filters by artists feelings and also shows us the lyrics of the songs, so it is very useful. On the other side, we find a radar of emotions, which shows the emotions that the songs give us. In such a way that we can also see by artist which ones have more emotions of a given type.

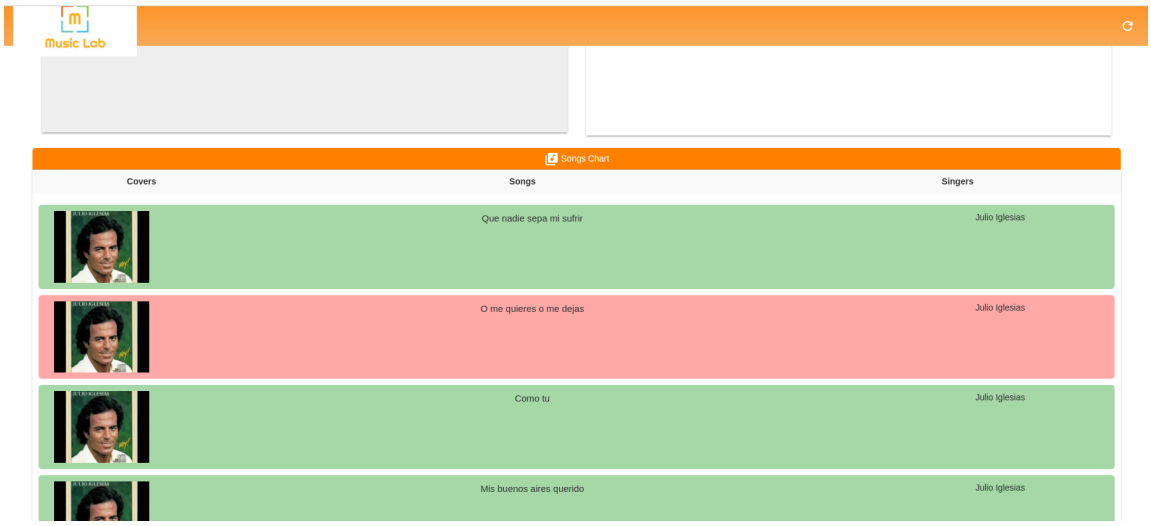

Figure 4.4: Songs tab part three

Finally we show the song chart again which we think is still very useful, since the visibility of the songs is always of great help. Also as a result of the filter in other widgets, we can see reflected the letters which we are interested.

4.6 Conclusions

In this last chapter, we present the steps we have followed for data analysis. In the same way we have also explained the usefulness of our website, how it can be used and the different widgets and views used.

In summary, we present the results of the analysis, observing that the artists whose songs emit a more positive feeling are Joaquín Sabina and Extremoduro, while, on the other hand, those who have more negative songs are Michael Jackson and Julio Iglesias. In the same way, emotions are shared about the artists. So regardless of the genre we can conclude that the songs represent their own feelings and emotions, since we chose four different groups of artists to be able to compare them.

CHAPTER 5

Conclusions and future work

5.1 Introduction

In this last chapter, we discuss the conclusions, achievements, problems and possible future work on our thesis.

5.2 Conclusions

In the project, we have created a board for the analysis of the emotions and feelings of the lyric songs based on Web Components, in this way we make the data visual and can interact with them. The board is formed by widgets to clearly show the data.

This project consists of five modules. The first is the search system that is responsible for collecting all the data of the songs in a json file. The second module is the orchestration system, responsible for building the pipeline by executing tasks and dependencies sequentially. The analysis system is the third module responsible for performing the analysis of these data by feelings and emotions. The fourth module is the indexing system responsible for collecting the data analyzed in our database. And finally the visualization system that is responsible for making all this readable to users, and therefore one of the most important

parts.

The following shows the problems we have faced, the objectives achieved and possible future work.

5.3 Problems faced

When carrying out our project, not everything went as we expected, then we name the most important problems that we face:

Space and installation: We chose to do our project on Linux in a Virtual machine, to which we assigned from the beginning a space in the virtual disk, which later turned out to be insufficient, so we had to do a disk expansion. On the other hand, the installation of Sefarad in virtual machines can not be done through the Docker [1] [7] containers, so it had to be done manually. For this same reason safarad boot has to be done manually, as was done in previous versions of sefarad.

Creation of the json file: When we made the creation of the Json file that was introduced in the orchestrator, we realized that this needed to have an identifier, so in the creation script of the Json an identifier was created that assigned a random number to each song between one and a billion with a probability of repeating numbers barely existing.

5.4 Achieved goals

The following section analyzes the results obtained and objectives achieved in the development of the project:

Design and implementation of a song extraction system: The first and important objective that we face and without which the project would not make sense, without the data of the songs, it would be impossible to advance in its development. In this way we chose Musixmatch as the tool capable of achieving this end, considering it as the most appropriate to collect the data we needed.

[1] https://www.docker.com/

Form a pipeline for the analysis of sentimens and emotions: After obtaining the data from the song extraction system, we needed a system that would organize the tasks of collecting the Json file, perform the analysis of feelings and emotions and index it in Elatisearch. To achieve the goal we chose Luigi as the task orchestrator. And Elasticsearch and Senpy as tools to achieve the objectives of storage and analysis of the songs respectively.

Design of an interactive dashboard: In order that all the analyzed data can be interpreted by the users, it is necessary to create a visualization system, this objective has been achieved by creating an interactive dashboard with different views and widgets through Polymer Web Components and the library D3.js.

Evaluate the objectivity of the singers: The last objective of the project achieved thanks to the previous steps. Using the analyzed data and the board created to show the different perspectives that compare sentiments and emotions in the different lyrics of the songs and artists.

5.5 Future work

Although this project has been developed as much as possible, improvements or features can always be made to the project.

Add more songs and artists: This could be the main issue to be addressed in the future. It would be useful to analyze the songs that we think may be more relevant to our case, that is, we would have more data to be able to extract more data and obtain better results.

Add new widgets: A dashboard has been built with certain widgets. More widgets from Polymer Web Components and D3.js could be built to show more information about the data.

Add new features: The project has been focused on the lyrics of the songs and their feelings and emotions. However something similar could be done semantically analyzing the words that are repeated in the songs, the most relevant, if they are adjectives, nouns, etc. And an evolutionary comparison over time of the lyrics of the songs in an artist.

Bibliography

[1] Mike Bostock. D3.js - data-driven documents, October 2017.

[2] G. Vulcu C. A. Iglesias, J. F. S'anchez-Rada and P. Buitelaar. "linked data models for sentiment and emotion analysis in social networks," in sentiment analysis in social networks., October 2016.

[3] Zachary Tong Clinton Gormley. *"Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and analytics engine"*. O'Reilly Media, January 2015.

[4] Enrique Conde Sánchez. "development of a social media monitoring system based on elasticsearch and web components technologies", Master's thesis, ETSI Telecomunicación, June 2016.

[5] Giuseppe Costantino, Francesco Delfino, Gianluca Delli Carri and Massimo Ciociola. Musixmatch, October 2017.

[6] I. Corcuera-Platas J. F. Sánchez-Rada, C. A. Iglesias and O. 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), October 2016.

[7] Adrian Mouat. *"Using Docker: Developing and Deploying Software with Containers"*. December 2015.

[8] Scott Murray. *"Interactive Data Visualization for the Web, 2nd Edition"*. O'Reilly Media, August 2017.

[9] Jarrod Overson and Jason Strimpel. *"Developing Web Components: UI from jQuery to Polymer"*. February 2015.

[10] Spotify. "Luigi Python module".