

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 DEVELOPER
SENTIMENT ANALYSIS SYSTEM BASED ON GITHUB
COMMIT COMMENTS**

ADRIÁN GONZÁLEZ HERNANDO
2019

TRABAJO DE FIN DE GRADO

Título: Diseño y desarrollo de un sistema de análisis de sentimientos basado en los comentarios de commits de GitHub.

Título (inglés): Design and development of a developer sentiment analysis system based on GitHub commit comments.

Autor: Adrián González Hernando.

Tutor: Carlos A. Iglesias Fernández.

Departamento: Departamento de 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
DEVELOPER SENTIMENT ANALYSIS
SYSTEM BASED ON GITHUB COMMIT
COMMENTS**

ADRIÁN GONZÁLEZ HERNANDO

2019

Resumen

El objetivo de este trabajo es llevar a cabo un sistema capaz de obtener comentarios de commits de GitHub para su posterior análisis de sentimientos.

En primer lugar, como es lógico, se ha necesitado una gran cantidad de comentarios de commits para poder finalmente establecer unas rigurosas conclusiones. Para ello, se han utilizado múltiples herramientas, descritas en la memoria, como son GHTorrent (a través de su página web o de mongoDB) o una desarrollada por mí a través de Python, haciendo peticiones a la API de GitHub.

A continuación, una vez que se han obtenido los datos, éstos han sido posteriormente analizados a través de Senpy y después el resultado del análisis con sus correspondientes datos (el repositorio en el que se ha hecho el commit comment, la hora a la que se ha creado...) han sido almacenados en Elasticsearch.

Una vez que los datos han sido almacenados en Elasticsearch, se ha podido proceder finalmente a la visualización de los resultados creando un cuadro de mandos por medio de Sefarad y Web Components, los cuales ofrecen una buena interfaz de usuario.

Posteriormente se podrán visualizar todos los elementos en el cuadro de mandos. Esta tesis permite al usuario extraer sus propias conclusiones sobre los sentimientos y los lenguajes de programación entre otros aspectos.

Palabras clave: Sefarad, mongoDB, Luigi, Scrap, Web Components, GitHub, Senpy, Elasticsearch, Luigi, commit comments, GHTorrent

Abstract

The objective of this end of degree project has been to develop a system that enables it to obtain commit comments from GitHub in order to be analyzed as sentiment.

First of all the procedure needs to gather a large number of commit comments to be able to draw some accurate conclusions. To pursue this task, we have used multiple tools, each of them has been described in this study. They are GHTorrent (through its website or mongoDB) and another one developed by myself through Python and with requests to the API GitHub.

Once the data has been obtained, it has been analyzed through Senpy. Later the result including its corresponding data has been stored using Elasticsearch. This data is formed by the repository name in which the commit comment has been made, the time they had been issued...

After the data has been stored in Elasticsearch, it is finally possible to visualize the results by creating a dashboard by using Sefarad and Web Components, which offers a good interface for the user.

Once all the elements can be visualized in the dashboard, these findings allow the user to draw his own conclusions about sentiments and programming languages as well as other features.

Keywords: Sefarad, mongoDB, Luigi, Scrap, Web Components, GitHub, Senpy, Elasticsearch, Luigi, commit comments, GHTorrent

Agradecimientos

Este Trabajo Fin de Grado supone para mí la finalización de otro ciclo de mi vida. Sin lugar a dudas, uno de los que más esfuerzo ha requerido.

En primer lugar quisiera dedicárselo a mis padres, Esther y Luciano, sin los cuales hubiese sido completamente imposible seguir este camino. Gracias a vosotros he podido realizar este trabajo, este grado y todo lo que ha conllevado, en definitiva, gracias por guiarme en este camino y en los que están por llegar. También se lo quisiera dedicar a la memoria de mi abuelo Emilio, por ser un ejemplo de constancia y superación.

En segundo lugar me gustaría agradecerse a mis amigos por ser otro apoyo fundamental, especialmente a Javi y Pablo.

En tercer lugar gracias a los miembros del GSI, por ayudarme en todo momento.

Por último y no menos importante, a mi tutor Carlos A. Iglesias por ayudarme a buscar el TFG idóneo, orientarme durante la realización del mismo y guiarme hasta concluir el proyecto.

Contents

Resumen	I
Abstract	III
Agradecimientos	V
Contents	VII
List of Figures	XI
List of Tables	XIII
1 Introduction	1
1.1 Context	1
1.2 Project goals and tasks	3
1.3 Structure of this document	4
2 Enabling Technologies	5
2.1 Introduction	5
2.2 GHTorrent	5
2.3 API REST GitHub	6
2.4 MongoDB	6
2.5 Senpy	7
2.6 Luigi	8
2.7 Elasticsearch	9

2.8	Sefarad	10
2.9	Polymer	10
3	Requirement Analysis	13
3.1	Introduction	13
3.2	Use cases	14
3.2.1	System actors	14
3.3	System sequence	15
3.3.1	Admin case	15
3.3.2	User case	16
3.4	Class	17
3.5	State diagram	18
4	Architecture	19
4.1	Introduction	19
4.1.1	Overview	19
4.2	First module: Obtaining data	20
4.3	Second module: Analyzing data	23
4.4	Third module: Indexing data	25
4.5	Fourth module: Visualizing data	26
5	Case study & Results	31
5.1	Introduction	31
5.2	Case Study	31
5.2.1	Repositories analyzed	32
5.2.2	Dashboard functionality	32
5.2.3	Elasticsearch requests	34
5.2.4	Dashboard update	36

5.3	Results	36
5.3.1	Programming language	36
5.3.2	Time	37
5.3.3	Date	38
6	Conclusions and future work	41
6.1	Introduction	41
6.2	Achieved goals	42
6.3	Future work	42
6.4	Problems overcome	43
6.5	Conclusions	44
	Appendix A Impact of this project	i
A.1	Introduction	i
A.2	Social impact	i
A.3	Environmental impact	ii
A.4	Economic impact	ii
A.5	Ethical Implications	ii
	Appendix B Economic budget	iii
B.1	Human resources	iii
B.2	Physical resources	iii
B.3	Licences & Taxes	iv
	Bibliography	v

List of Figures

2.1	Senpy processes	7
2.2	Luigi's diagram	8
2.3	Sefarad architecture	10
3.1	Use case UML diagram	14
3.2	Sequence admin UML diagram	16
3.3	Sequence UML diagram	17
3.4	Class UML diagram	18
3.5	State diagram	18
4.1	Architecture	20
4.2	GHTorrent's output	22
4.3	First module	23
4.4	On the top Google chart Bar-chart, on the bottom Google chart pie	27
4.5	Number chart	28
4.6	Material search chart	28
4.7	Commit comment chart	29
5.1	Dashboard implementation	32
5.2	Senticommit	33
5.3	Commit comment's distribution per programming language	37

List of Tables

1.1	Sentiments per programming language	2
1.2	Sentiments per repository	2
1.3	Commits Sentiment Analysis Statistics	3
5.1	Repositories	32
5.2	commit comment's distribution per time	38
5.3	Commit comments per date	38

Introduction

1.1 Context

Every day human beings can experience different feelings in the various aspects of life: in our private life, at work or in any other possible situation. All these feelings can be sensed by human beings as long as people are interacting with each other. We can feel if a person is frustrated due to the fact of having to face a negative situation as well as we can see if somebody is experiencing a state of enjoyment. Sentiments can affect employees' performance at work, especially programmers.

Happiness at work has traditionally been seen as a potential by-product of positive outcome at work, rather than a pathway to business success. On the one hand, in the labor market programmers do not choose what projects they would like to develop, and in the long run, their interest might decrease and negatively affect their work. On the other hand, non-professional programmers can develop what they would like, without the need of delivery deadlines or pressures from their superiors. Anyway, this is not an excuse to acknowledge that programming can be frustrating.

These feelings can be analyzed by software capable of detecting emotions, opinions or assessments. How society feels about a specific topic can be easily detected by the

most popular social networks such as Facebook or Twitter. This thesis is going to analyze sentiments in GitHub.

A similar analysis to this one has already been carried out in different projects and their outcomes will be compared with those in this thesis. Their findings will be dealt with bellow. The first research shows how much positive/neutral/negative the programming language programmers are. Its conclusions are shown in the next table.

	Positive	Neutral	Negative
Ruby	51%	17%	32%
Javascript	43%	20%	37%
c#	41%	20%	38%
Python	39%	20%	40%
Java	39%	19%	41%
PHP	38%	18%	44%
C	36%	21%	42%
C++	33%	21%	46%

Table 1.1: Sentiments per programming language
<https://github.com/sAbakumoff/gh-comments-sentiment>

The following survey [1] analyzes commit comments of certain repositories. Some of these repositories have also been used in this thesis in order to compare both results. The results of this study can be observed in the following Table 1.2.

	Jquery	Rails	CraftBukkit	Diaspora	MaNGOS	TrinityCore
Positive	0.30	0.36	0.25	0.32	0.35	0.30
Neutral	0.31	0.33	0.28	0.37	0.31	0.40
Negative	0.39	0.31	0.47	0.31	0.34	0.30

Table 1.2: Sentiments per repository

Although the upcoming research [2] (its results are in the Table 1.3) is not about commit comments, it deals with sentiment analysis of GitHub commits. The author questioned the following “are the security comments or discussions different (sentiment-wise) than the rest of the comments or discussions?”. The conclusion was, as seen in Table 1.3, that there are more negative discussions than commits.

Type		Negative	Neutral	Positive
Discussions	Security	75.52 %	10.88%	16.58%
	Rest	54.28%	20.37%	25.33%
Comments	Security	55.59%	23.42%	20.97%
	Rest	46.94%	26.58%	26.47%

Table 1.3: Commits Sentiment Analysis Statistics

<https://www.researchgate.net> Security and emotion: Sentiment analysis of security discussions on GitHub

In this thesis, first we have to get the data from GitHub. GitHub provides a REST API that allows us to get the data. At first the tool that was used to extract data is going to be GHTorrent and finally it was a personalized scraper. GHTorrent and the scraper obtain data from Github public event time line. GitHub REST API provides users names, commits, commit comments, pull requests. As it mentioned above, commit comments are going to be the tool for the analysis of the sentiments.

Once the commit comments pass the Senpy filter (sentiment analysis) the result will be saved in Elasticsearch. Finally the results will be shown in a dashboard.

1.2 Project goals and tasks

The purpose of this project is to classify the comments that people upload on the Internet, especially the GitHub commit comments that programmers upload.

The tasks of this project are the following:

- To work with GHTorrent obtaining the GitHub data through its REST API.
- To filter this data with Senpy Analyses in order to obtain the classification of feelings.
- To utilize these data with Elasticsearch.

- To present such data using Sefarad.

1.3 Structure of this document

This section focuses on the structure of this document will be explained, which is made up of 6 chapters.

Chapter 1 This chapter provides a global vision of the project.

Chapter 2 This chapter describes all the tools and technologies that have been used.

Chapter 3 This chapter shows different UML graphs and schemes that have been considered important in order to explain the analysis and design of the project.

Chapter 4 This chapter is about all the modules that make up the architecture.

Chapter 5 This chapter deals with a case study in greater depth and also the results will be detailed and analyzed.

Chapter 6 This chapter tries to share the conclusions that have been obtained as a result of this thesis.

Enabling Technologies

2.1 Introduction

This chapter briefly explains the different technologies involved in the whole project. Starting at the moment when the commit comments are scrapped to finally reach their visualization. The most important technologies are: Sepy, Elasticsearch, Sefarad, Web Components and GitHub REST API.

2.2 GHTorrent

GHTorrent¹ is a project that gets the information from the GitHub REST API. Therefore, GHTorrent can be used to obtain data (users, commits, ids..) from GitHub. It is possible to collect all kinds of GitHub information of a certain day. The information provided is classified by the most important aspects of GitHub that are: commit comments, commits, events, followers, forks, pull requests, repositories, users...

GHTorrent gives us the chance of downloading this data as a MySQL database or

¹<https://github.com/gousiosg/github-mirror/>

MongoDB database.

2.3 API REST GitHub

The API REST GitHub² provides all kind of information from GitHub: issues, repositories, users, code... A REST (Representational State Transfer) is a web service. REST is any interface between systems that uses HTTP to obtain data or generate operations on that data in all possible formats, such as XML and JSON. These operations can be: POST, GET, PUT, DELETE, HEAD or OPTIONS.

This service will be used to scraping the commit comments of a specific user and repository. In the chapter 4.2 will be explained the structure of the request.

2.4 MongoDB

MongoDB³, from humongous, is an open source non relational data base program which is programmed in C++. MongoDB allows us to the read information given from GHTorrent by indexing it in mongoDB.

MongoDB can save any type of data in BSON file that is the binary representation of JSON, therefore it allows us to export the information in JSON format. As mongoDB is non relational database, predefined schemes are not required. It uses JSON structures in the same way that SQL uses the database table to save the information. MongoDB comes with a standard console from which it can be executed the different commands (export data, import data, create users...). This console is built on JavaScript, so queries are made using that language. All this characteristics makes mongoDB highly scalable, flexible.

MongoDB gives us the possibility to use it in localhost and in the cloud service. MongoDB is used in multinationals like Telefonica, SEGA, or coinbase.

Finally mongoDB is used to index documents but also to search and store data.

As it will be mentioned later, mongoddb's operation is based on database and collections.

²<https://api.github.com/>

³<https://www.mongodb.com/>

2.5 Senpy

Senpy⁴ [3] is an open-source software for text analysis (sentiment and emotion) developed by the Intelligent Systems Group (GSI) at the ETSIT-UPM. Senpy employ semantic vocabularies (e.g. NIF, Marl, Onyx) and formats (turtle, JSON-LD, N-Triples and a graph) what is an important advantage in order to collect information.

Senpy has nine different plugins by default which can be tested in the web. These plugins analyze text in Spanish and/or English. There are two kinds of plugins.

The first kind of plugins that Senpy offers, analyze the text sentiment whose outputs are positive, negative or neutral. Secondly there are plugins that analyze the emotions whose outputs can be happy, sad, angry... Some of the most important are:

- *sentiment-basic* is a sentiment classifier that uses rule-based classification.
- *emotion-aneu* is an emotion classifier that uses ANEW lexicon dictionary to calculate valence-arousal-dominance of the sentence. Once we have VAD value emotion-aneu classified the text provided into the emotion that has the closer VAD value.

The framework consists of two main components:

- Senpy core is the building block of the service.
- Senpy plugins consist of the analysis algorithm.

The following figure explains Senpy's text analysis process.

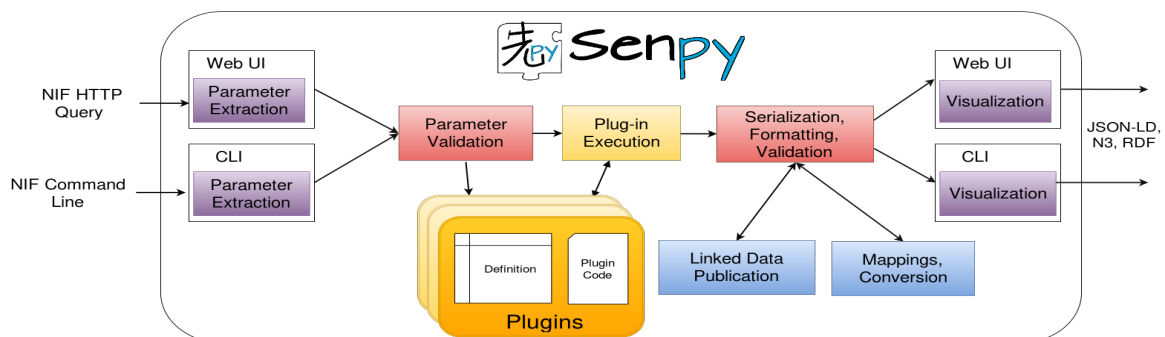


Figure 2.1: Senpy processes

⁴<https://senpy.readthedocs.io/en/latest/>

2.6 Luigi

Luigi [4] is a Python package developed by Spotify in 2014. Luigi was created in order to facilitate the creation or the management of complex pipelines. A pipeline is a set of elements in the way that the output of an 'n' element is the input of the 'n+1' element. This allows an effective management. Luigi reduces the probability of failure due to it starting the process with previous results if a problem occurs.

Luigi is very easy to implement in this project due to its simplicity. It is made up by four building blocks: target, task, parameters and dependencies.

1. *Target*: Target is a file that you have and it is the start-point. Although this may be necessary, in practice it is not always so.
2. *Task*: Task is where computation is done.
3. *Parameters*: Parameters is such as a constructor for each luigi Task.
4. *Dependencies*: As Luigi use tasks, objectives and parameters, Luigi allows you to express arbitrary dependencies in code instead of using some kind of uncomfortable DSL configuration (dependencies are often very complicated).

Task is where computation is done. The Luigi methods that alter the behavior are: `run()`, `output()` and `requires()`. During execution process, Luigi checks for errors and if there has been any, it will stop the execution of the pipeline. We can observe the data flow in the Figure 2.2.

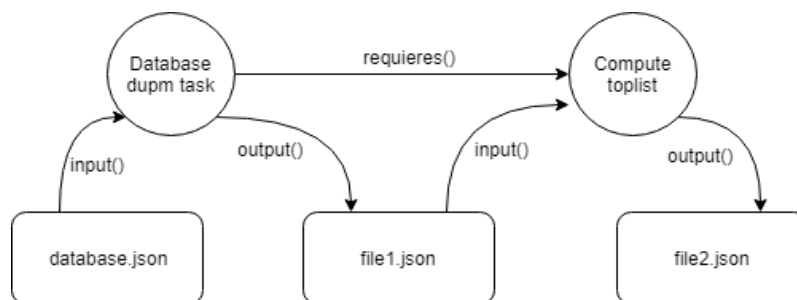


Figure 2.2: Luigi's diagram

In this thesis, Luigi is used to create and manage a pipeline that obtains and analyzes the commit comments and also indexes data into Elasticsearch. Chapter 4.3 will explain how a LuigiTask works.

2.7 Elasticsearch

Elasticsearch [5] is an open-source, RESTful, distributed search and analytics engine built on Apache Lucene. It is developed in Java. Elasticsearch, Logstash and Kibana⁵ are known as ELK. ELK is a data manager that has a distributed storage in Elasticsearch. The information is processed by Logstash and the information is displayed in Kibana and queries can be made in real time. If it is necessary to use a large amount of different data types it would be recommended to use ELK. However, as this thesis does not need different data types (just a large amount of commit comments) we are not going to use ELK. Then we will only use Elasticsearch in order to store data.

Elasticsearch allows us to store data in JSON documents. It adds an index, later the data can be searched or retrieved by using the Elasticsearch API.

There are basic parts of Elasticsearch that are used.

- *Cluster*: A cluster is a set of one or more nodes that keep all the information distributed and indexed.
- *Node*: A node is a server which is a component of a cluster, stores your information and helps with the tasks of indexing and searching the cluster.
- *Index*: It is a collection of documents that has similar characteristics. The indexes will be used for indexing, searching, updating and deleting information into Elasticsearch. Indexes are identified by a name.
- *Sharding* and *Replicas*: These tools are designed in order to make the indexing task easier.

Elasticsearch is used because more than 2,000 comments are going to be analyzed and Elasticsearch offers a great response for both indexing and removing tests or erroneous files. It also uses JSON objects as an answer, therefore the information given can be managed better.

⁵<https://www.elastic.co/es/elk-stack>

2.8 Sefarad

Sefarad⁶ is an environment in charge of visualizing data and making SPARQL Protocol and RDF Query Language queries. SPARQL is a standardized language for the consultation of RDF graphs. Sefarad has also been developed by the GSI-UPM. As it will be explained below in chapter 4.5, Sefarad is connected to Elasticsearch to be able to represent the result of the analysis of the commit comments through web components. These web components are based on Polymer.

Sefarad architecture is the following:

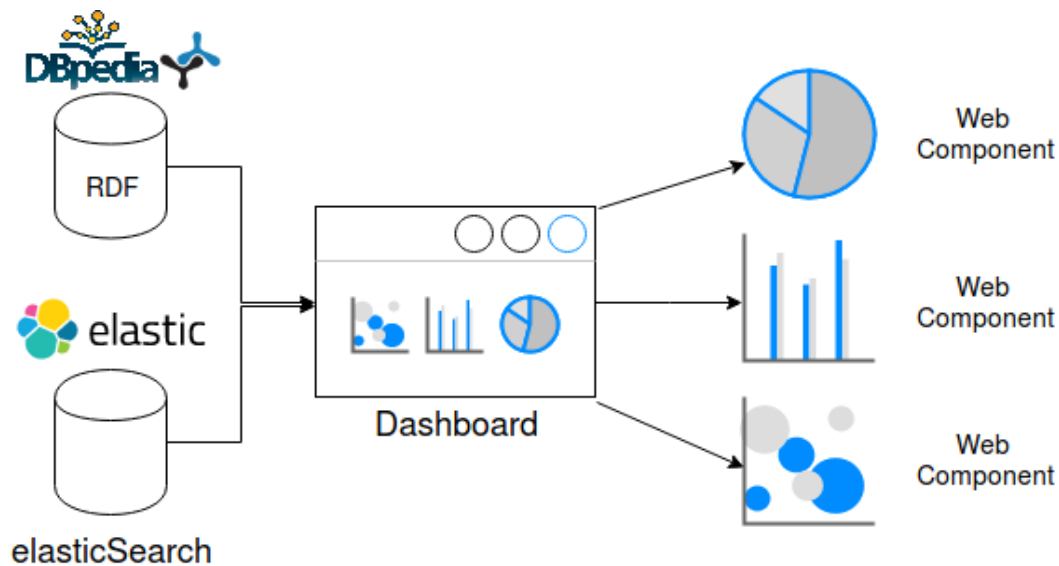


Figure 2.3: Sefarad architecture

To sum up, Sefarad is based on two main modules. The first is in charge of the visualization and the other one is in charge of Elasticsearch.

2.9 Polymer

Polymer is an open-source JavaScript library, developed by Google and GitHub users, that use web components for making web applications.⁷ The main browsers supports web components except Edge that they are developing it⁸. There are many multinationals that use Polymer for their web services such as Google (YouTube, Google Play Music.), Netflix,

⁶<https://sefarad.readthedocs.io/en/latest/>

⁷[https://en.wikipedia.org/wiki/Polymer_\(library\)](https://en.wikipedia.org/wiki/Polymer_(library))

⁸<https://www.webcomponents.org/>

BBVA, IBM...

Polymer allows users to make custom elements for each user. The Polymer web components use the following technologies: HTML, CSS and JS. These programming languages are the most popular ones for front ends, what makes Polymer web components being appealing. They are also interactive making them a good option for the user to interact.

Requirement Analysis

3.1 Introduction

This chapter describes the software using UML software system through Unified Modeling Language diagrams. UML diagrams is a standard language for representing any kind of system. UML allows us to visualize, specify, construct and document a software system.

UML diagrams [6] are important because a graphic in software engineering can represent a software system to collect those aspects of relevance.

The UML diagrams that are going to be used in this thesis are: use cases, sequence, class and state diagrams. Thanks to these diagrams we will be able to get an understanding of how the application works.

To sum up, this chapter is not going to go in depth explaining all modules or the full system. It will only depict a brief idea of how the system works.

3.2 Use cases

Use case diagrams explain the main system uses. The limitations that each user has in the application are going to be explained as well as the tasks that each user can fulfill. The external systems that are involved in each task are also going to be explained.

3.2.1 System actors

The following figure represent the system actors diagram.

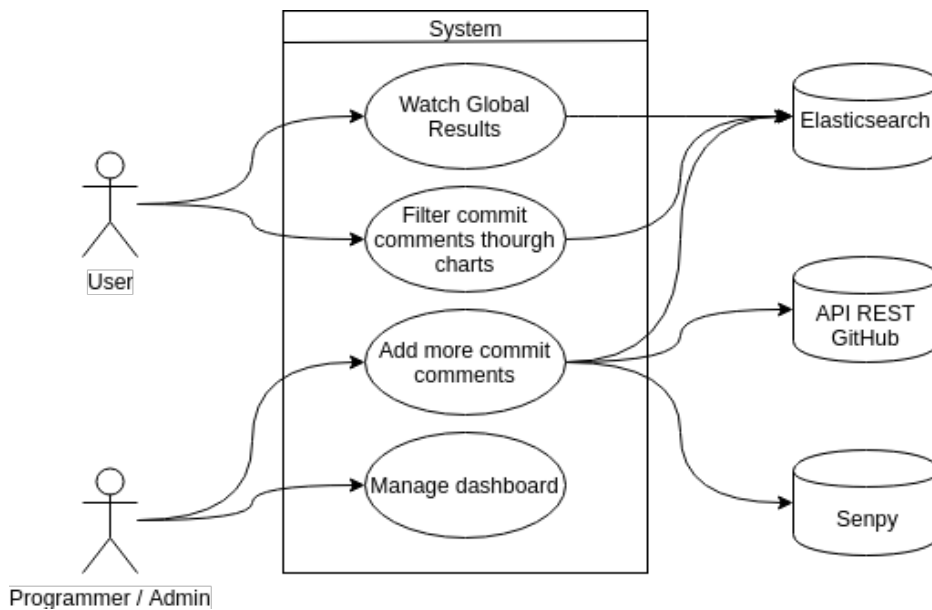


Figure 3.1: Use case UML diagram

We can observe that there are two main actors, four uses and three secondary actors are involved in the system.

The actors and the system characteristics are:

- *User*: The system is focused on the user, this is the main actor. The user can manage the Dashboard, he can also deal with the interactive charts and can draw their own conclusions. These interactions are treated as filters, in which the user can select a specific characteristic of the commit comments such as the programming language, the repository or other, this way the user obtains the sentiments of these commit comments.
- *Programmer / Admin*: The admin is allowed to introduce more commit comments in

order to enrich the content of the system. The admin can also introduce more charts or modify whatever he wants.

The secondary actors characteristics are:

- *Elasticsearch*: In a nutshell, Elasticsearch is where the data to be provided to the user are stored. If the admin wants to introduce more data, it will be necessary to upload it to Elasticsearch. In Chapter 4.4 it will be explained in depth.
- *API REST GitHub*: The commit comments are scraped from API REST GitHub.
- *Senpy*: The tool that analyses the commit comments.

3.3 System sequence

In this section a system sequence diagram (SSD) will be explained. A system sequence diagram shows how the different external tools interact with the system in order to get the final result.

Two cases are going to be explained through a system sequence diagram: the admin and the user cases.

3.3.1 Admin case

As it can be seen in Figure 3.5, there are four external tools: API REST GitHub, Senpy, Elasticsearch and Web components.

- *API REST GitHub*: The commit comments are scraped from API REST GitHub. Not all data provided by GitHub is used. In order to analyze commit comments the required parameters are: 'html_url' (the url of the commit comment), 'repository' (the repository where the commit comments lie), 'log in' (the user log in that owns the repository), 'id_user' (the log in of the user that has done the commit comment), 'date' (the date that the commit comment has been created) and the last one 'body' (the commit comment).
- *Senpy*: The parameter that GitHub provides as a 'body' is given to Senpy in order to be analyzed. Senpy provides the result of a commit comment in JSON-LD format. It is used to classified the commit comments.

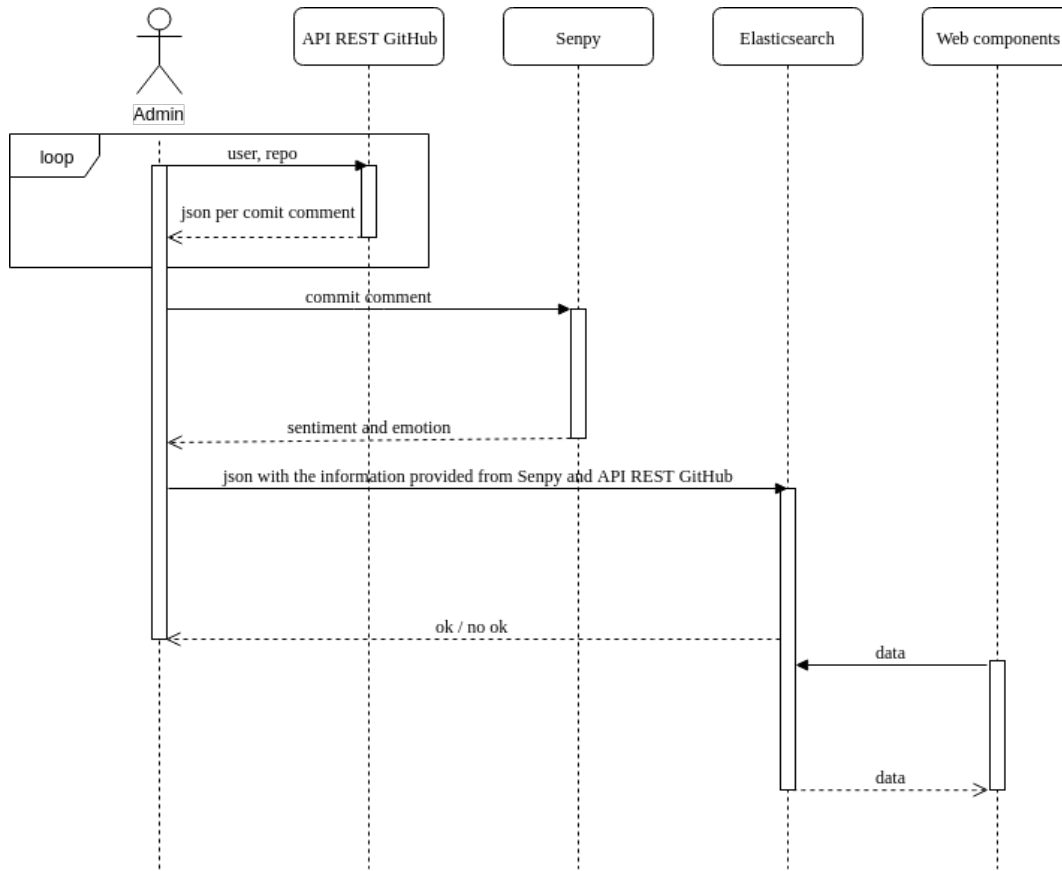


Figure 3.2: Sequence admin UML diagram

- *Elasticsearch*: Once the data has been analyzed, it needs to be stored in Elasticsearch. Elasticsearch needs a JSON with the information, an index and a doc-type.
- *Web components*: Web components request Elasticsearch for data. Once the components have obtained the data they can represent it. The request takes in account the user's preferences.

3.3.2 User case

This case is formed by two external modules: Web components and Elasticsearch.

- *Web components*: The user is able to interact with web components. As previously stated, the user can filter the commit comments so as to be able to show a specific kind of commit comment. When a user selects a specific characteristic, the web components take the commit comments from Elasticsearch with the chosen characteristics. Then they are represented through web components.

- *Elasticsearch*: It provides data to web components.

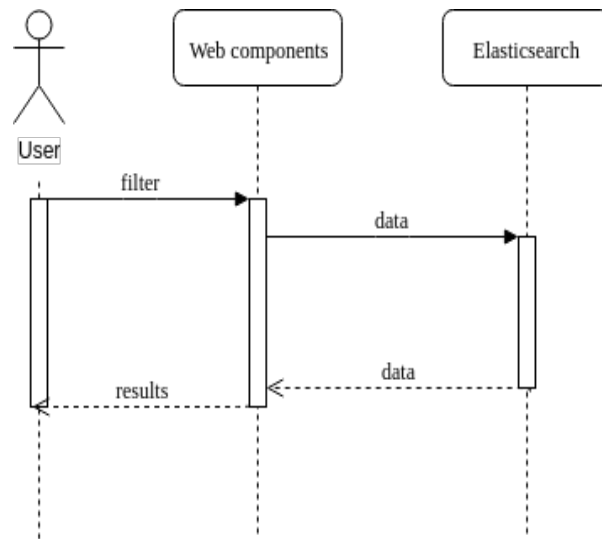


Figure 3.3: Sequence UML diagram

3.4 Class

The last UML diagram that is used is the class diagram. A class diagram describes the structure of a system by showing the system's classes with all its characteristics (attributes, methods and their needs).

As it can be seen in the UML diagram, we are able to see at a glance all the dependencies and classes that the system needs. There are 3 classes: `ScrapyTask()`, `AnalyzeTask()` and `Elasticsearch()`. These three classes will be explained in detail in the upcoming Chapter 4.

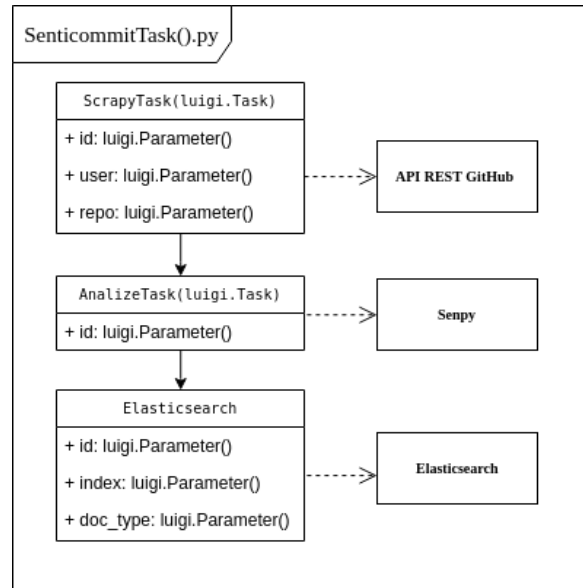


Figure 3.4: Class UML diagram

3.5 State diagram

The application has two main states:

1. The first state is that the user can observe the global results of the sentiment analysis. At the same time, the user will be able to interact with the charts in order to proceed to the following state or to end the request.
2. The second state is the display of specific results once a filter has been applied to the charts. In addition, the user can select another filter as many times as possible. The user will be able to return to the previous state if he wishes to apply another filter or just to see the global ones. The user can also end his query from this state.

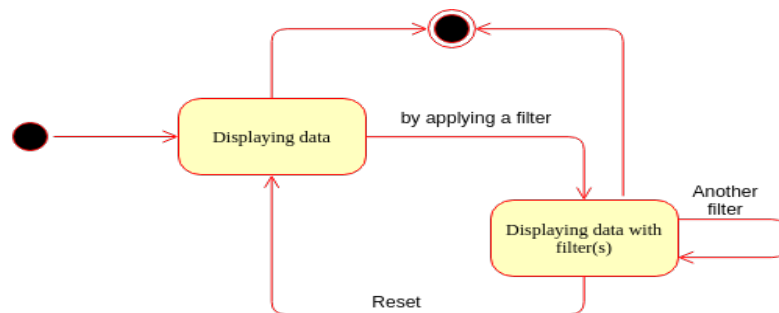


Figure 3.5: State diagram

Architecture

4.1 Introduction

This chapter presents the different phases of the project. First of all, a basic structure of the project is presented so that we can get an idea of the different modules of the project. Once we have a general view, explaining how the different modules have been created, developed and designed.

4.1.1 Overview

This section is going to focus on a general view of the project. The thesis is made up by four different modules, which will be further explained more in detail as follows. The four modules are: obtaining, analyzing, indexing and visualizing data.

1. The first module is composed by GHTorrent, mongoDB and Python. There will be two ways of obtaining data. The first with GHTorrent and mongoDB and the second through a Python class. As previously stated in Chapter 2.2, GHTorrent is in charge of capturing data of the GitHub REST API where commit comments will be obtained. The Python class is a LuigiTask in charge of capturing data from a specif repository.

2. The second module captures data provided by Senpy. Senpy returns the sentiment of a text that has been previously provided to Senpy through Luigi, a Python package.
3. Then the third module is in charge of indexing data into Elasticsearch through a Python class. Luigi is also used so as to create pipeline only for uploading data.
4. Finally the last module is in charge of representing graphically via web components, the results provided by Senpy and the data that was capture from module one such as the user-log in, the commit comment, etc. It explains how to make a Dashboard based on web components through Sefarad and Polymer and how it gets the data from Elasticsearch.

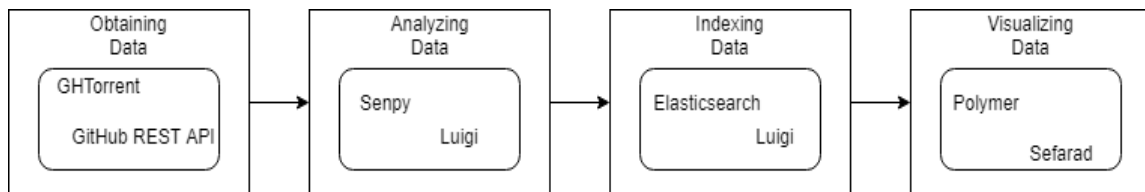


Figure 4.1: Architecture

4.2 First module: Obtaining data

There are three ways that have been used in this thesis in order to obtain commit comments from GitHub.

The first one is using GHTorrent¹. On GHTorrent website we can download² the information of GitHub referring to a given day. Therefore this way has been used in order to get a great deal of commit comments regardless the importance of the repository that it is going to be analyze.

Next the data have to be indexed in mongoDB, because the information given from GHTorrent's website is in BSON format, so that we can visualize all the information through mongoDB.

In order to be able to index the documents, a database has to be created. There are three configured databases (local, admin and test) by default. Databases hold collections of documents, in other words, mongoDB stores documents in collections contained in databases. Once a database has been created or a default one is going to be used, a collection must be

¹<https://github.com/gousiosg/github-mirror/>

²<http://ghtorrent.org/downloads.html>

established in order to index the document. After we set up a collection and a database, it can be imported into mongoDB an 'input.bson' file through console.

Once it has been verified that the data is correct, it has to be exported in JSON format because Elasticsearch needs this information in JSON format as it will be seen in the next module. It can be exported by the following command via console.

```
$ mongoexport --db <database-name> --collection <collection-name> --out  
output.json
```

The other options to get information is to get data from a specific repository. Two different ways to obtain this information are going to be shown as well as the reason why one of them should be chosen.

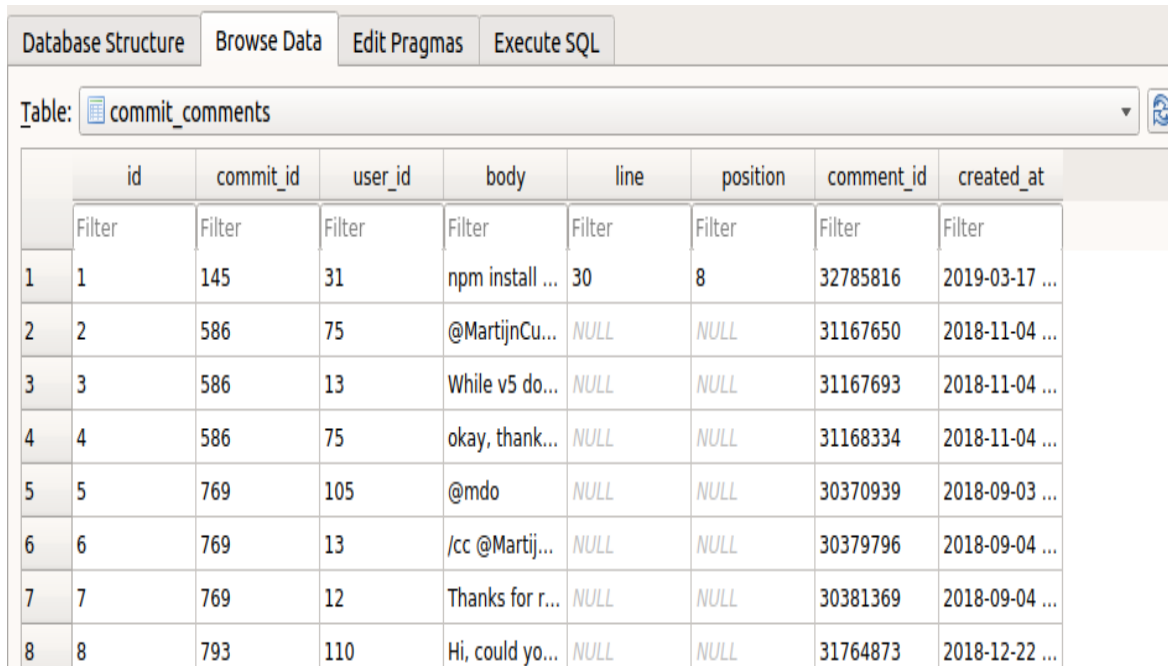
- The first possibility is through a Python method using Luigi which will be explained in the next module, in a nutshell this class³ only needs the user's name who owns the repository and the repository name. This Luigi class retrieves the information given from the GitHub REST API through the following URL (in which x means any number): <https://api.github.com/repos/USER/REPOSITORY/comments/comments?page=X>. GitHub REST API offers 30 commit comments per page. If we focus on a single commit comment GitHub REST API shows the information related to a commit comment in a JSON format such as the log in of the user who made the commit comment, the link, the commit comment, etc.. Therefore, the commit comments will be saved in a JSON file for further analysis.
- The second possibility is through GHTorrent and its entire database. Once all Gems⁴ required by the program are installed, GHTorrent gives us also the chance to retrieve all the information related to a repository or a single user/organization through two different commands. As this thesis will be based on the study of particular repositories, only the first option would be used. This option can be executed through the following command:

```
$ ruby -Ilib bin/ght-retrieve-repo user repository
```

The final result that GHTorrent provides us is a file.db which can be converted to JSON. Figure 4.2 shows an output example. It can be seen that the fields returned from GitHub are *commit_id*, *user_id*, *body*, *comment_id* and *created_at*.

³It is a Luigi class.

⁴<https://guides.rubygems.org/what-is-a-gem/>



	id	commit_id	user_id	body	line	position	comment_id	created_at
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	145	31	npm install ...	30	8	32785816	2019-03-17 ...
2	2	586	75	@MartijnCu...	NULL	NULL	31167650	2018-11-04 ...
3	3	586	13	While v5 do...	NULL	NULL	31167693	2018-11-04 ...
4	4	586	75	okay, thank...	NULL	NULL	31168334	2018-11-04 ...
5	5	769	105	@mdo	NULL	NULL	30370939	2018-09-03 ...
6	6	769	13	/cc @Martij...	NULL	NULL	30379796	2018-09-04 ...
7	7	769	12	Thanks for r...	NULL	NULL	30381369	2018-09-04 ...
8	8	793	110	Hi, could yo...	NULL	NULL	31764873	2018-12-22 ...

Figure 4.2: GHTorrent's output

As far as this thesis is concerned the first method does not satisfy the requirements of this thesis cause it retrieves data from every repository. While the third method has been ruled out because obtaining data and analysis in the same Python file is much more simpler. Furthermore the only useful data that provides GHTorrents are the fields: 'created_at' and the target 'body'. This thesis is going to make a more complete study therefore the programming language or the user (the one who did the commit comment) are required in that database. Although all methods have been used during the developing of the project.

To sum up, we can comprise this section in the following figure in which dbX means a database and colY means a collection. If we use the first method the scheme would correspond to the top of the diagram. The method that has been used has a gray background. As it can be seen, using any method of the three commented it is able to get the target commit comments and continue with the following second module.

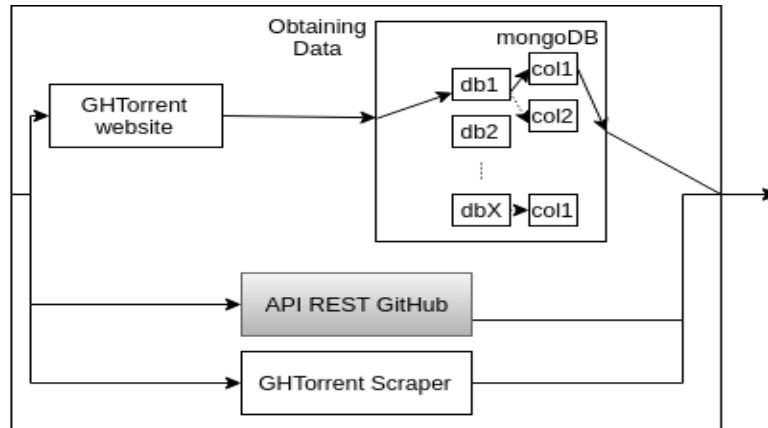


Figure 4.3: First module

4.3 Second module: Analyzing data

After exporting the data if first or third method has been used in the first module, now it is necessary to use Python. If second method has been used, this module is programmed after the previous task with other Luigi Tasks. Then it will be explained in detail how the tasks of this Python file work.

The Python file contains the methods and classes that are going to be charge of reading the data as well as extracting as stated above, analyzing via Sefarad and storing them via Elasticsearch. Luigi is used so as to facilitate the tasks of the pipeline (a chain of processing elements).

The task of analyzing data will now be explained in detail. The task of obtaining data would be practically the same.

A Luigi class can be called `MyTask(luigi.Task)`. Luigi, as previously stated, needs some methods to work such as `run()`, `output()`, `requires()` or `input()`.

- To analyze through Luigi and Senpy, the `requires()` method is needed because the `AnalyzeTask` needs to start working with the JSON file that it is retrieved from `ObtainingDataTask`. So `ObtainingDataTask` will be called by `AnalyzeTask` so that `AnalyzeTask` can get the required output. Therefore `AnalyzeTask` proportionate the parameters that the `ObtainingDataTask` needs. Whereas if the third method from the module 1 is used, `requires()` should be empty because `AnalyzeTask` would already have the data provided by GHTorrent.
- The `run()` method contains the code that is going to be run. It is in charge of reading

the JSON obtained from the module 1. Once the Luigi Task is able to read the JSON file, the Senpy analyze is used. The commit comment is introduced in Senpy in order to be analyzed by choosing a suitable filter. Then this information will be given in a JSON-LD file. JSON-LD⁵ seeks the least possible effort from developers to transform their existing JSON into JSON-LD. It is not used to validate JSON data. The parameter which is capture from Senpy is 'marl:hasOpinion '. The attribute 'marl:hasOpinion' provides the attributes 'marl:hasPolarity' that shows the sentiment of the commit comment that has been submitted to the analysis.

This information besides the id and the user of GitHub, which come from module 1, are stored in the output().

- Finally the output() method defines where data is going to be stored. It can be created another JSON file with the information provided from Senpy and some parameters from the module 1 (user, repository, commit comment...) that will be upload it in Elasticsearch so that later the dashboard can access these data to represent them. This will be explained in detail in the upcoming third module (Indexing data 4.4).
- Another Task that it is important to mentioned but it is not used in this thesis is the input() method. It is a wrapper around Task.requires that returns a specific Target objects.

In the following chart it is possible to visualize a Luigi's structure. The chart illustrates how the Luigi class and its methods are defined, as well as the 'main' at the end of the chart.

```
class MyTask(luigi.Task):  
    # Parameters for this task  
    param = luigi.Parameter(default=42)  
  
    # What other tasks it depends on  
    def requires(self):  
        return SomeOtherTask(self.param)  
  
    # The business logic of the task  
    def run(self):  
        f = self.output().open('w')  
        print >>f, "hello world"  
        f.close()  
  
    # Where it writes the output
```

⁵<https://en.wikipedia.org/wiki/JSON-LD>

```
def output(self):  
    return luigi.LocalTarget('/tmp/foo/bar-%s' % self.param)  
  
if __name__ == '__main__':  
    luigi.run()
```

4.4 Third module: Indexing data

This module tries to explain the tasks involved in uploading data to Elasticsearch. First of all it is necessary to understand how Elasticsearch works. As stated above section 2.7, Elasticsearch needs an 'index' and a 'doc-type' they are not the only ones it needs.

There are few more parameters that need to be explained.

- *id*: it has been mentioned before, remember that it has to be unique.
- *index* and *doc-type*: they have also been mentioned before.
- *host* and *port*: it is where Elasticsearch is going to be deployed.
- *timeout*: as its name suggests it is a timeout in case of an error.

Then it will now be explained how to upload JSON to Elasticsearch. There are at least two different methods for uploading files into Elasticsearch. The first is through the console which is very similar to the deleting which is going to be explained later. The other one is through Luigi. It is much better because it is in the same pipeline as the second module, in other words, it indexes it automatically.

Luigi is also used for uploading data. You can notice in the box below the structure of the Elasticsearch Luigi Class.

```
class Elasticsearch(CopyToIndex):  
  
    id = luigi.Parameter(default= today)  
    index = luigi.Parameter()  
    doc_type = luigi.Parameter()  
    host = 'localhost'  
    port = 9200  
    timeout = 100  
  
    settings = {"index.mapping.total_fields.limit":6000, "number_of_shards":  
                1, "number_of_replicas": 1}
```

```
def requires(self):  
    return OtherTask(self.id)
```

Once the file that is going to be uploaded it is ready, the only command needed to upload the file is as follows. This command only needs as parameters the index and the doc-type.

```
$ python file.py Elasticsearch --index index --doc-type doc-type --local-  
scheduler
```

If for example we have a file (a JSON) uploaded and we need to delete it because it has some error that we have not noticed, the file can be removed with the following command⁶.

```
$ curl -X DELETE 'http://localhost:9200/index/doc_type/X'
```

4.5 Fourth module: Visualizing data

Once the previous processes are successfully finished, data that is stored in Elasticsearch, must be presented in order to be visualized creating a dashboard.

The way that this thesis presents data is through Sefarad. Sefarad uses Polymer, a JavaScript library (as it is mentioned in Chapter 2.9) that allows to use web components in which we can classified data under any given criteria. The dashboard is configured with the index and the doc_type that Elasticsearch used so as to get the results of the data that a web component needs to work (total of commit comments, the result of the analysis, etc). Each web component needs different parameters which will be explained bellow. This parameters are given as aggregations from Elasticsearch.

On the following chapter (chapter 5) it will be explained the structure of the data in Elasticsearch and how these aggregations are produced.

To sum up, the dashboard is composed by some web components. The parameters that these needs to work are different from each other and are ordered from Elasticsearch as aggregations.

The web components used in this thesis are following.

⁶<https://www.elastic.co/guide/en/elasticsearch/reference/6.6/getting-started-explore-data.html>

- **Google chart Bar-char:** This chart is obtained thanks to Google Chart API ⁷. It lets the user to visualize data in a bar-chart. In this thesis, this chart is used to visualize how many commit comments are neutral negative or positive. The parameters of this web components are:
 - *data*: It requires a JSON with the information provided from Elasticsearch.
 - *field*: The field requires a field to visualize.
 - *type*: It is the Google chart type.
 - *filters*: The filters requires a field to visualize.
 - *cols*: It indicates on the abscissas' axis what each column means.
- **Google chart pie:** This widget is also obtained thanks to Google Chart API ⁸. Therefore it is similar to the previous one but there is a difference between them and this is that as its own name indicates, Google chart pie is a pie chart.

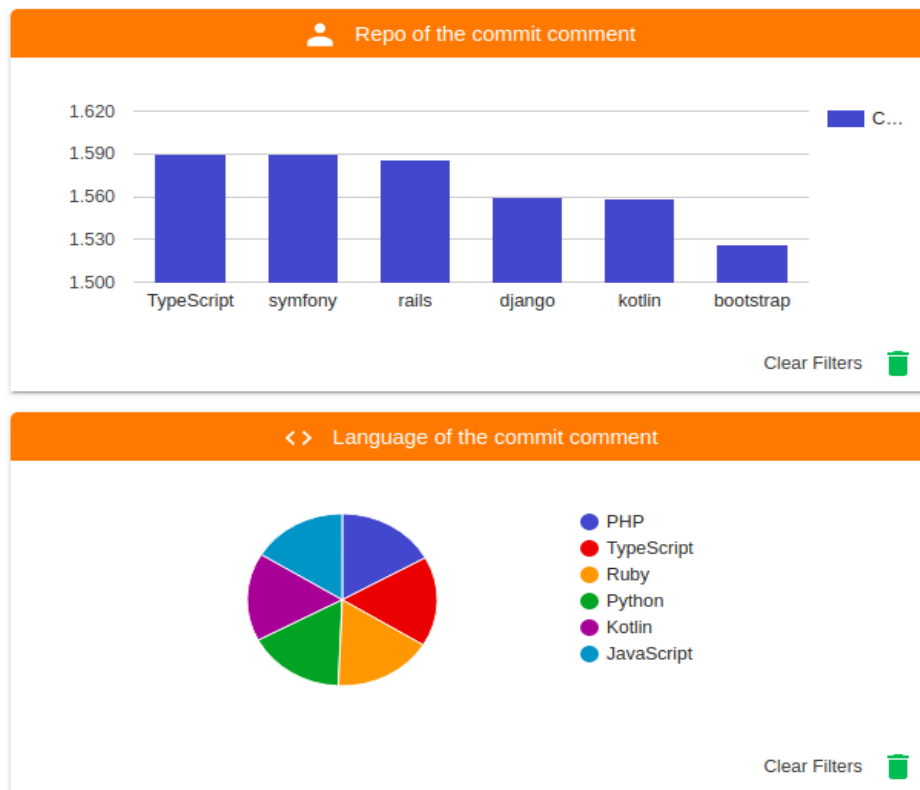


Figure 4.4: On the top Google chart Bar-chart, on the bottom Google chart pie

⁷<https://developers.google.com/chart/interactive/docs/gallery/columnchart>

⁸<https://developers.google.com/chart/interactive/docs/gallery/piechart>

- **Number chart:** These widgets let the user visualize data in a different way from the one before mentioned. This web component stores how many elements there are of a given type and also gets the total of elements of all types. It can be appreciated in the Figure 4.5) that the total of the commit comments are 9.4k and there are 4.2k 'Afternoon commit comments'. This web component is used to classify commit comments by parts of the day: morning, afternoon, evening and early morning. The parameters that are needed for this web components are:
 - *data*: The data requires a JSON with the information provided from Elastic-search.
 - *title*: It sets the title of the web component.
 - *aggkey*: The aggkey requires the aggregation name in data.
 - *object*: It need the value of the field to represent.
 - *stylebg*: It indicates widget background.

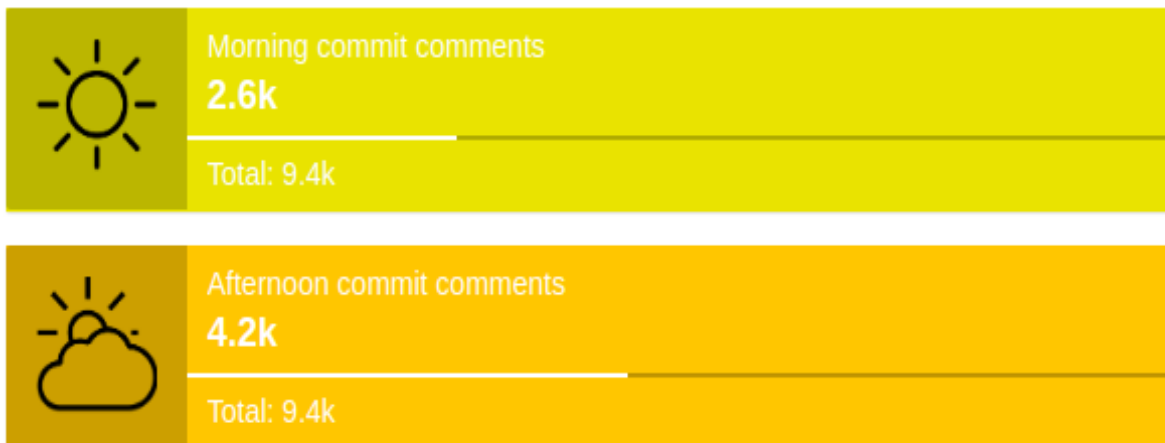


Figure 4.5: Number chart

- **Material search chart:** This web component is a search tool designed to make queries to other web components. The query parameter is automatically updated with the search box. This web component does not need any type of parameter.



Figure 4.6: Material search chart

- **Commit comment chart:** This web component lets the user visualize a list of the commit comments that have been analyzed and also through its green, grey or

red background it can be seen if it is a positive, neutral or negative commit comment respectively. The parameters that it needs are:

- *data*: The data requires a JSON with the information provided from Elastic-search.
- *face*: It sets an icon in the title of the widget.



Figure 4.7: Commit comment chart

Case study & Results

5.1 Introduction

This chapter explains a use case in detail and we analyze the results that we have obtained through the analysis of sentiments. The first thing we do is explaining how the sequence is carried out when the user interacts with an element of the dashboard. In the second place the results of the analysis of sentiments are shown divided into: programming language, repository, time...

5.2 Case Study

The users' use case is now introduced. We talk about which repositories have been chosen for the analysis, also how the web components of the dashboard work and finally we deal with the requests for Elasticsearch.

5.2.1 Repositories analyzed

First of all it is necessary to specify which repositories have been chosen. These repositories can be viewed in the following table.

Language	PHP	TypeScript	Ruby	Python	Kotlin	JavaScript
User/Repository	symfony/symfony	microsoft/typescript	rails/rails	django/django	JetBrains/kotlin	twbs/bootstrap

Table 5.1: Repositories

These repositories have been chosen for several reasons.

- The first reason is due to their importance and popularity. All of them are among the three with most stars, which is equivalent to giving it a like in any other social network.
- Secondly because they have a large number of commit comments to make the analysis representative, more than 1,500 each and some even more than 2,000. Although we have tried to choose a similar figure among them to draw conclusions more easily.
- Thirdly, these languages have been chosen in order to be able to compare these results with the other studies commented on the chapter.

5.2.2 Dashboard functionality

The dashboard, as mentioned above, is composed of web components. The dashboard implementation can be seen below.

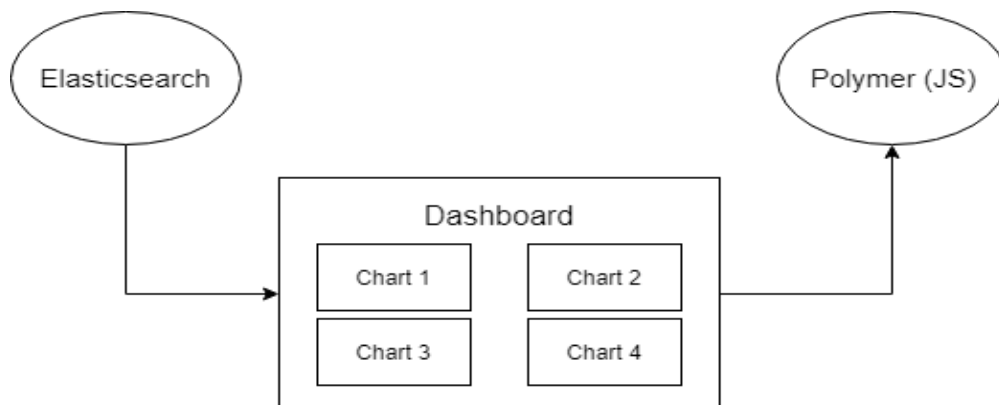


Figure 5.1: Dashboard implementation

It is possible to visualize the dashboard with the web components in any browser through Polymer. It provides the charts with all their needs. Polymer gives a set of polyfills that allows us to use web components in browsers that are not compatible with an user-friendly framework. Polyfills [7] emulate the browser capabilities that are missing as closely as possible.

Once the user enters into the application the user will find the following interface (Figure 5.2) composed by the web components that have been mentioned and described in chapter 4.5.

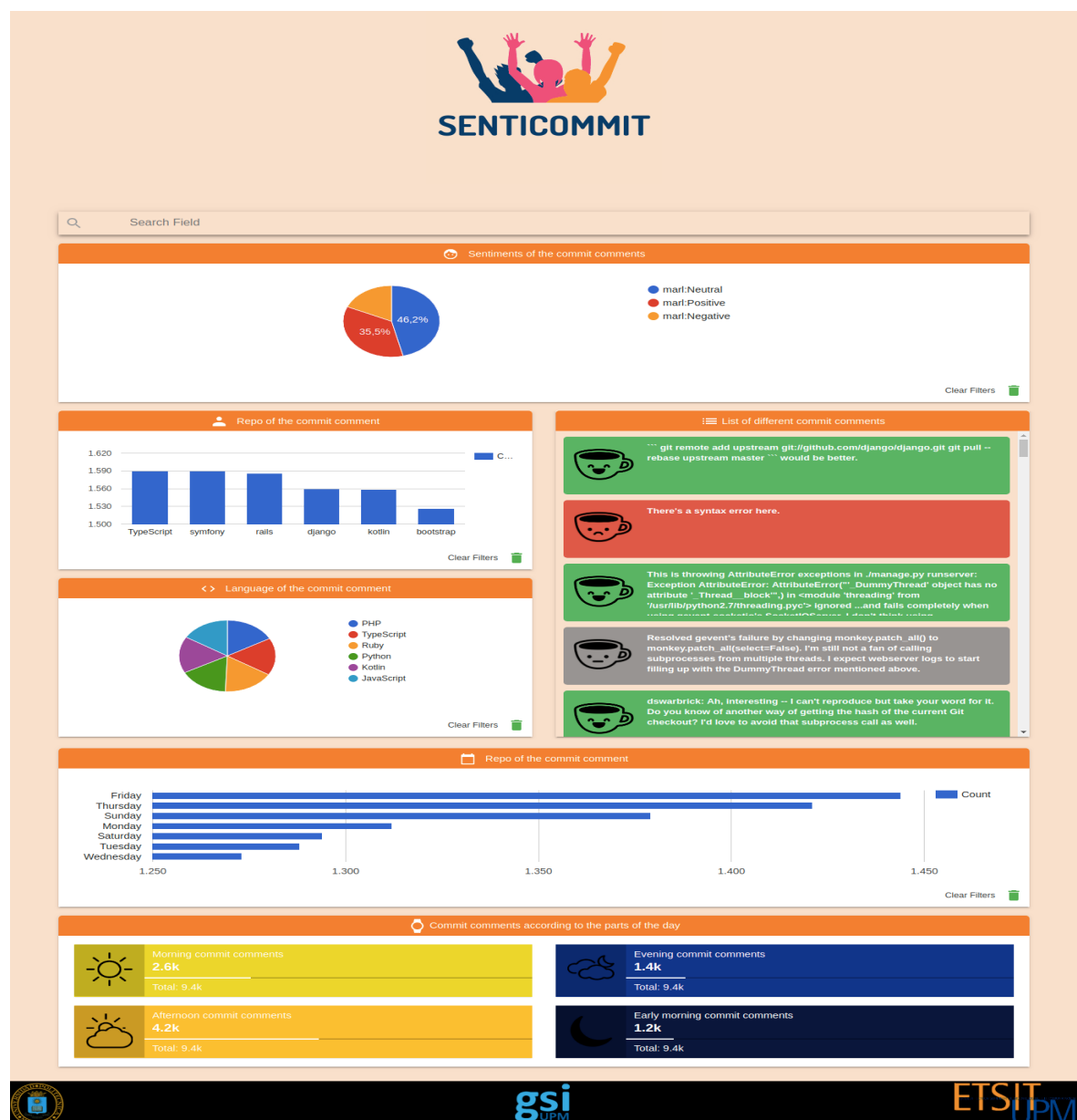


Figure 5.2: Senticommit

By clicking on a filter of the graphics, as mentioned in previous chapters, we will be able to obtain commit comment with the requested requirements. In other words, if for example a repository is selected, all commit comments for that repository will be obtained and the rest of the filters will be updated according to the new parameters. This fact will give rise to the petition of Elasticsearch.

5.2.3 Elasticsearch requests

This section will describe the procedure by which the system performs the requests to obtain certain commit comments chosen by the user.

The web component, in the case of the example the Google pie chart, will send the request to Elasticsearch.

The Elasticsearch structure must be explained before the aggregation structure is shown with the commit index. These fields have been chosen because they have been considered essential. If a different type of fields were to be obtained, they could be added. The different fields of Elasticsearch are the following:

- *id_user*: The *id_user* field indicates the user who has made the commit comment and has been saved for a future use case, for example to create a chart in the Dashboard with the top ten most positive users.
- *repo*: The *repo* field indicates the repository where the commit comment is located. Thanks to this field we can count how many commit comments belong to a certain repository.
- *language*: The *language* field indicates the language.
- *commit*: The *commit* field indicates the body of the commit comment, i.e. the message
- *sentiment*: The *sentiment* field is formed in turn by a set of three values, including a JSON-LD. The field 'marl:hasPolarity' indicates the sentiment of the commit comment.
- *partsDay*: The *partsDay* field indicates the phase of the day to which the commit comment belongs, i.e. whether it was issued in the morning in the afternoon etc.
- *date*: The *date* field indicates the day on which the commit comment was created.

Once the structure of the commit index in Elasticsearch is known, proceed to continue with the example mentioned above. If the request is made with a specific repository, a petition like the following would be made.

```
GET commit/_search
{
  "size": 0,
  "aggs": {
    "repo": {
      "terms": {
        "field": "repo.keyword",
        "min_doc_count": 1,
        "size": 10
      }
    }
  },
  ,
```

Therefore once Elasticsearch obtains the request it will return a result like the following.

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 0,
    "max_score" : null,
    "hits" : []
  },
  "aggregations" : {
    "repo" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "bootstrap",
          "doc_count" : 1558
        }
      ]
    }
  }
}
```

Once the answer is obtained, the dashboard is updated.

5.2.4 Dashboard update

Once the user has selected a specific feature and the response has been obtained from Elasticsearch, the web components are updated. Therefore the web components will show the commit comments data containing this feature.

For example in the case of bootstrap, the Google pie-chart of the dashboard (Figure 5.2) will show the relationship between the sentiments of the bootstrap's commit comments, the commit chart will show the bootstrap's commit comments and so on with all the web components.

If the user wishes to apply another filter, let us consider an example such as selecting 'marl:Positive' in the Google pie-chart, the process would be repeated.

5.3 Results

Next we will present the results of the feelings shown in the confirmation comments according to other parameters such as the programming language, the day and the time.

5.3.1 Programming language

The proportion of sentiments can be seen in Figure 5.3:

It can be noticed how most of them are neutral in the different programming languages. The reason for this case is that many commit comments just look for a technical explanation of the repository and others just do not show sentiments.

Knowing that 'Rails' has been Ruby's repository chosen for analysis and comparing the results with the other studies we can demonstrate several things.

- The programming languages of Kotlin and TypeScript have not been analyzed in any of the studies mentioned but in this one it has, What is the reason? The answer is simple. These programming languages have grown according to GitHub by 260% and 190% respectively¹. Due to the importance that these languages will have in the upcoming future, they have been considered relevant enough to be taken into account in this project.
- The JavaScript and Ruby languages, bootstrap and rails repositories respectively, the

¹<https://giveitaspin.net/>

results obtained in this thesis are very similar to those obtained in other projects.

- The results of Java analysis have been quite different. The reason may be due to the fact that the repositories chosen have been different.

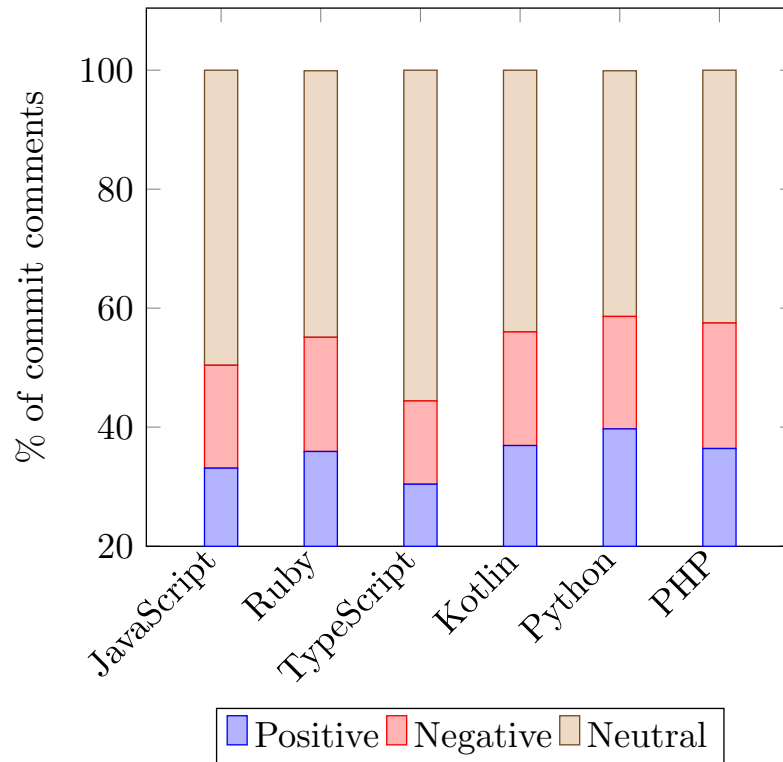


Figure 5.3: Commit comment's distribution per programming language

To sum up, the obtained results have been quite similar to the ones of other studies mentioned in this thesis. This fact confirms the accuracy of the results.

5.3.2 Time

The commit comments have also been grouped by the time they were made. The results of the time analysis will be discussed below.

In order to draw a relationship between commit comments and time, commit comments have been divided into four sections: morning, afternoon, evening and early morning. The hours between intervals have been allocated to each of these periods:

Morning	Afternoon	Evening	Early morning
[6,13)	[13,21)	[21,0)	[0,6)

Table 5.2: commit comment's distribution per time

The distribution of commit comments throughout the day is as follows. As it could not be otherwise, more commit comments take place in the afternoon reaching 44.68%. Secondly in the morning 27.66%. Finally the evening and early morning commit comments are the least frequent reaching 14.89% and 12.77% respectively.

Once that the repositories and the commit comments have been analyzed, it has not been possible to get a very accurate relationship between sentiments and hours. The only scenario with clear and precise conclusions is the early morning one. The numbers of positive feelings between morning, afternoon and night are around 36%, whereas the early morning period only reaches 31.8%.

5.3.3 Date

Regarding the analysis on a day by day bases we should prioritize the days in which the highest or the lowest amount of commit comments have been produced.

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Commit Comments	13.94%	13.68%	14.52%	15.09%	15.34%	13.75%	13.68%

Table 5.3: Commit comments per date

Taking into account that the sample space is seven, every day of the week, it is significant the difference that commit comments made between Friday and Tuesday.

With regard to sentiments, the most significant findings are the following.

- Friday is when the most negative commit comments and the fewest positive commit comments are issued. Regarding the total commits made on Friday, the negatives commit comments represent 20.77% and the positives 33.37%.
- Regarding the total of the commits carried out on Thursday, it is significant that on this day we obtain the lowest number of negative commit comments and the highest

number of neutrals. On the one hand, it is interesting that only 17.03% of the commit comments made on Thursdays are negative. On the other hand, neutrals account for 47.43%.

- Saturday is when there are more positive commit comments and less neutral. Of the total commit comments made on Saturday 37.10% are positive, whereas 44.20% are neutral.
- The rest of the days are kept at constant and similar values. Positive commit comments represent about 35.5%, negatives about 18% and neutrals about 46%.

Conclusions and future work

6.1 Introduction

A GitHub commit comments sentiment analyzer has been developed in this project. In order to carry out this system a good organization has been needed, which can be well differentiated in several modules.

The first thing that was carried out was the scraper, which obtained the data of the repositories that were specified. After all the data from the different scrapers had been obtained, the commit comments were analyzed and then our own data was created. Having obtained all the data, it was stored and classified. Then the visualization of these data has been carried out implementing a dashboard, which requests the stored data whenever the user needs it. Finally, the project concludes after the data has been presented and the users will be able to draw their own conclusions.

6.2 Achieved goals

In this work towards the final degree several tools have been presented with the purpose of measuring the sentiments observed among computer developers.

- First of all, GitHub API data retrieval tools such as GHTorrent have been used. This tool can be used to obtain any data from GitHub.
- During project development the use of mongoDB was also required as part of GHTorrent. The use of a NoSQL database was required, as well as the creation and use of specific user profiles for these databases and collections.
- For the main focus of the project concerns data was extracted, they had to pass a sentiment filter like Senpy's one. Once all results were obtained, including Senpy's, they had to be stored in an Elasticsearch index for later comparison. These tasks were made possible using Luigi.
- Finally the last aim of the project is the development of a dashboard which is able to show the results stored in Elasticsearch and also the ability to be interactive upon by a user. The dashboard, which was developed by Sefarad, is based on Polymer Web Components. These components are the charts which offer the user the versatility needed by dashboard. This results in an interface that is interactive and intuitive towards the needs of the user.

6.3 Future work

The tasks that would remain pending in this thesis are:

- One of the pending options with this project would be to be able to make an analysis of a specific segment of the programmers, that is to say, to be able to take the commit comments from the same user/company and see what results could be obtained, having the same sample of people for the different repositories. This way we can obtain more accurate results of the feelings about the programming languages. However, there is currently no user/company in GitHub that has a large number of repositories with many commit comments in order to be representative.
- Adding more web components to the dashboard to make it even more visual, for example with a field-chart, so you can see the sentiments of the commit comments according to the time.

- Another pending task is getting more repositories so that we can do a more complete analysis of programming languages such as: C, R or HTML.
- Finally, another task that could be done in the future is to create a filter of emotions.

6.4 Problems overcome

The problems I have faced in this thesis are multiple. The first one was when I accepted the job, because I was not familiar with any of the technologies I have finally used. However, if some problems need to be highlighted, there is no doubt that the main problems that have slowed down the process of job creation have been the following:

- The installation of GHTorrent was much more complicated than it really is. As I said before, my lack of knowledge of both mongoDB and GHTorrent and in addition not knowing Ruby language, caused a big delay in the project.
- Another problem to point out was the installation of mongoDB. Although the official mongoDB documentation is explained step by step, the latest version caused Broken count > 0 and pkgProblemResolver Errors. This was a nightmare as nothing could be installed or removed from the computer because these errors were constantly there. This bug could not be fixed by any of the following recurring commands:

```
$ sudo apt-get upgrade && sudo apt-get update
$ sudo apt-get install -f
$ dpkg --configure -a
```

Finally dependencies of packages that were badly installed from /var/lib/dpkg/status were removed (the package was mongodb-org-tools), ignoring the problems this would lead to.

- Another problem was the limited user requests to the GitHub API, only 60 per hour. If you exceed 60 requests per hour, you are banned for one hour without being able to make any additional requests. This situation when scraping was a problem, as it frequently exceeded these 60 requests.

6.5 Conclusions

In this end-of-degree project we have accomplished a design and development system that extracts commit comments, analyzes and exposes them through the different technologies hereby described. Furthermore, the commit comments have been classified according to several criteria: programming language, repository, time and day of the week in which the commit comments were published. As many as 9,500 commit comments have been analyzed, and even though it is a large amount of commit comments, they are not enough to establish rigorous conclusions. Nevertheless, certain indications can be withdrawn. The sentiments can vary a great deal from one programming language to another, also between repositories of the same language. Moreover as it has been previously mentioned, it can vary according to the day. This is mainly due to the type of audience: back-end or front-end. In general, the languages that can be used for front-end tend to be more negative than those used exclusively for back-end. Finally, if the system could be developed with an even larger number of commit comments, it would be possible to establish more rigorous conclusions for this case, as well as enabling us to establish a direct relationship between hours and sentiments. That has been previously stated.

Impact of this project

A.1 Introduction

Studies of sentiments have always been recurrent when it comes to drawing conclusions from human behaviour. This is the case of emotions affecting our performance in sports [8] or how emotions affect learning and achievement [9].

This appendix explains the different impacts of commit comment sentiment analyses at a social, environmental, economic and ethical level.

A.2 Social impact

Sentiment analyses of commit comments have an important social impact. This thesis describes which programming languages are the most positive or negative as well as in which days and hours the most positive and negative commit comments are produced. Therefore, anyone who is thinking about what programming language to program a job could be benefited from this study. The quality and quantity of information of a programming language may be influenced by the amount of rejection or approval of that language.

A.3 Environmental impact

This section deals with the environmental impacts of this research.

Regarding environmental impacts the use of computers and computer screens must be considered. We should especially focus on the battery of the computer as the most polluting aspect of the tools used.

To sum up, it can be stated that this thesis does not have an excessive environmental impact.

A.4 Economic impact

This section deals with the economic impacts of this thesis.

The Introduction (chapter 1.1) shows how feelings affect the performance of workers in their jobs. Thus people who work happily reach a better job performance. This means that if workers' schedules are adjusted according to the hours in which they are at their happiest mood, costs could be saved for the company.

A.5 Ethical Implications

This section deals with the ethical impacts of this thesis.

The only relevant ethical aspect of this thesis is to take any commit comments from GitHub. This could be unethical because a chart of the dashboard publishes them. However the commit comments that are taken for analysis are totally public and anyone could access them. Furthermore, the commit comments that have been chosen for this analysis belong to GitHub's most popular repositories which means that they have already been widely exposed to the public.

Economic budget

This appendix details an adequate budget to bring about the project...

B.1 Human resources

This section deals with the different economic aspects of human resources used in this thesis.

As the project has been carried out by a person with no previous experience in the sector, the estimated salary is 12.5€ per hour. The necessary hours to carry out the project have been around 320 hours, therefore the total earnings is 4,000 €.

B.2 Physical resources

This section is mainly about exposing the devices used in the development of the project.

It consists basically of the personal computer and a second screen, although the screen is completely optional.

- **Computer:** Its features are: Ubuntu 18.04, 16GB RAM, Intel i7, 500GB HDD, 15.7".

The price is around 650€.

- **Second screen:** Its features are: IPS display, 24.2", anti-reflective. The price is around 250€.

B.3 Licences & Taxes

When it comes to licences, this project has been accomplished with Open Source Software (OSS), which means that there is no need to pay for the use of any of the technologies utilized.

Related to taxes we should take into account the Spanish legislation which establishes that a company must pay an extra 32.6% of the employee's salary. This amount breaks down as follows:

- 23.6% for common contingencies.
- 5.5% for unemployment.
- 3.5% for possible work-related accident.

Bibliography

- [1] Emitza Guzman, David Azócar, and Yang Li. Sentiment analysis of commit comments in github: An empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 352–355, New York, NY, USA, 2014. ACM.
- [2] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. Security and emotion: Sentiment analysis of security discussions on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 348–351, New York, NY, USA, 2014. ACM.
- [3] 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)*, pages 735–742, Montreal, Canada, October 2016. IEEE.
- [4] Erik Bernhardsson, Elias Freider, and Arash Rouhani. `spotify/luigi-github`.
- [5] Clinton Gormley and Zachary Tong. *Elasticsearch: The definitive guide: A distributed real-time search and analytics engine*. ” O’Reilly Media, Inc.”, 2015.
- [6] Yashwant Waykar. ”importance of uml diagrams in software development”. 03 2013.
- [7] K. Sons, C. Schlinkmann, F. Klein, D. Rubinstein, and P. Slusallek. `xml3d.js`: Architecture of a polyfill implementation of `xml3d`. In *2013 6th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, pages 17–24, March 2013.
- [8] Richard S Lazarus. How emotions influence performance in competitive sports. *The sport psychologist*, 14(3):229–252, 2000.
- [9] Reinhard Pekrun. The impact of emotions on learning and achievement: Towards a theory of cognitive/motivational mediators. *Applied Psychology*, 41(4):359–376, 1992.
- [10] Oscar Araque. Design and Implementation of an Event Rules Web Editor. Trabajo Fin de Grado, Universidad Politécnica de Madrid, ETSI Telecomunicación, July 2014.
- [11] J. Fernando Sánchez-Rada. Design and Implementation of an Agent Architecture Based on Web Hooks. Master’s thesis, ETSIT-UPM, 2012.
- [12] Munmun De Choudhury and Scott Counts. Understanding affect in the workplace via social media. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work, CSCW ’13*, pages 303–316, New York, NY, USA, 2013. ACM.

BIBLIOGRAPHY

- [13] Georgios Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.
- [14] Daniel Suárez. Design and development of a system for sleep disorder characterization using Social Media Mining. Trabajo Fin de Grado, ETSIT, Madrid, June 2018.