TRABAJO DE FIN DE GRADO

Título:	Desarrollo de un recolector de Social Media para análisis de sentimientos
Título (inglés):	Development of a Social Media crawler for Sentiment Analysis
Autor:	José Emilio Carmona López
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	Mercedes Garijo Ayestarán
Vocal:	Carlos A. Iglesias Fernández
Secretario:	Juan Fernando Sánchez Rada
Suplente:	Álvaro Carrera Barroso

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE GRADO

DEVELOPMENT OF A SOCIAL MEDIA CRAWLER FOR SENTIMENT ANALYSIS

José Emilio Carmona López

Febrero de 2016

Resumen

Este trabajo fin de grado contiene el resultado de un proyecto cuyos objetivos han sido diseñar y desarrollar los siguientes elementos:

- Un sistema de recopilación de comentarios en redes sociales y portales de opinión.
- GSI Crawler, un portal web que, utilizando el anterior sistema, recogerá y analizará los comentarios de los distintos portales.
- Implementación de un servicio para programar, monitorizar y administrar el sistema de recolección de comentarios.

Se describirá el desarrollo de los scrapers para la extracción de comentarios sociales. Se ha creado un scraper para cada portal web. Facebook, Twitter y Youtube ofrecen la información necesaria mediante el uso de una API propia. En cambio, Amazon, Yelp y TripAdvisor no ofrecían ninguna API para este fin, por lo que se ha tenido que construir un *scraper* a medida para cada uno de estos portales.

A continuación, se describirá el desarrollo del portal web GSI Crawler. Este portal web es una herramienta útil para el análisis de comentarios de cualquiera de estos portales antes mencionados. El usuario elegirá el tipo de análisis que quiere llevar a cabo (emociones, sentimientos o detección de comentarios falsos) y además facilitará, por ejemplo, un URL directo a un negocio de Yelp ó TripAdvisor. GSI Crawler descargará los comentarios pertenecientes a este elemento y, posteriormente, ejecutará el análisis pertinente usando la herramienta Senpy. Una vez el análisis haya finalizado, se mostrará un resumen del resultado y además se brindará la posibilidad de revisar uno a uno cada comentario junto con su calificación extraída del análisis.

Finalmente, se recogerán las conclusiones extraídas del proyecto, la tecnología que se ha aprendido durante el desarrollo del mismo y las posibles líneas de futuro trabajo.

Palabras clave: Analisis de sentimiento, scraper, Yelp, Twitter, Facebook, Youtube, Amazon, TripAdvisor, JavaScript, Polymer, Python

Abstract

This thesis collects the result of a project whose objective is to design and develop the next elements:

- A comments collection system for social networks and recommendations sites.
- GSI Crawler, a website that, using the previous system, will collect and analyze the comments from the different websites.
- Implementation of a service to schedule, monitor and administrate the crawling system.

It will be described the development of scrapers to collect comments. A scraper has been developed for each website. Facebook, Twitter and YouTube offer the necessary information through the use of a specific API. Otherwise, Amazon, Yelp and TripAdvisor don't offer an API which we could extract the comments, therefore a custom *scraper* has had to be developed to each one of these websites.

Next, the development of GSI Crawler will be described. This website is useful to the analysis of comments from any website mentioned before. The user will choose the type of analysis he wants to carry out (Emotions, Sentiments or Fake Analysis) and the user will also supply, for instance, a direct URL to a Yelp's Business, the *id* of a Facebook's Fan Page or a YouTube's Video. GSI Crawler will download the comments belonging to this element and, later, the pertinent analysis will be run using the Senpy tool. Once the analysis is finished, a summary of the result will be shown and the possibility of review each comment one by one will be also offered.

Finally, we gather the extracted conclusions from this project, the technologies we have learned during the development and the possible lines of future work.

Keywords: Sentiments analysis, scraper Yelp, Twitter, Facebook, Youtube, Amazon, TripAdvisor, JavaScript, Polymer, Python

Agradecimientos

- Al Grupo de Sistemas Inteligentes (GSI) por el apoyo durante la realización del proyecto así como el soporte con la herramienta Senpy para la implementación al proyecto.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Context

Sentiment Analysis [14] is a term used to talk about the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source information. In other words, Sentiment Analysis is used to look for the opinions in content and choosing the sentiment within those opinions.

The social repercussion of an institution, a business or, for instance, a new product launch could be measured by the amount of comments that users could make in the different social networks about these [4]. However, this way of repercussion measuring may be misleading as this way is not taking into account the importance of the comments' content and, simply, they are being treated like a number. For instance, a product could have many mentions in the social networks but it could be that most of these mentions could be negative.

For this reason, the sentiment analysis is a very important tool to take into account. A sentiment analysis system can analyze the comments content and it can give an overview more approximate to the users' opinion towards an institution, product

or business.

Nevertheless, a sentiment analysis system needs to be fed with the content to be analyzed. An analyzer must be combined with a scraper system to collect comments from the different social networks so that, together, they are able to obtain the expected result.

Most of these social networks offer a public interface called API (Application Programming Interface), with which the useful information could be gathered. Nevertheless, some interesting websites do not offer this public interface and the information should be extracted in a less trivial way.

The automated gathering of data from the Internet is nearly as old as the Internet itself. Although *Web Scraping* is not a new term, in years past the practice has been more commonly known as screen scraping, data mining, web harvesting, or similar variations.

Web Scraping [1], in theory, is the practice of gathering data through any means other than a program interacting with an API (or, obviously, through a human using a web browser). This is most commonly accomplished by writing an automated program that queries a web server, requests data (usually in the form of the HTML and other files that comprise web pages), and then parses that data to extract the needed information.

The modern Web is far from the characteristic stasis of the beginnings of the Web. Access to website contents can be restricted by means of authorization and authentication techniques. In addition, web pages are not longer static, and are based on a combination of client and server side dynamic rendering techniques.

This advantage of the modern Web might be a problem for the Web Scraping. When content is published through an API, a simple script can obtain interesting content calling that API. Nevertheless, when the content is only available at the presentation level, scraping techniques are needed, that could require the simulation of the behavior of a modern Web Browser.

In this final work, during the development of the scrapers, some problems have appeared in the gathering of the interest information and we have solved this inconveniences using several tools that enrich the scripts. The main function of some used tools, such as Selenium, is not the function we have done of these tools, but it has been a key factor to solve some of these inconveniences.

1.2 Project goals

The main goal of this project is to provide a platform for sentiment analysis for social networks and opinions websites. With this aim, the project will develop a system for collecting comments, a module to analyze these comments with Senpy tool and will develop a Website (GSI Crawler) that works like a user interface to run those analyses.

The previous main goal can be divided into the following sub goals:

- Design and build a scraper for several social networks (Twitter, Facebook, YouTube), for several opinions websites (Yelp, TripAdvisor) and for Amazon.
- Build a software module which is able to communicate with the Senpy platform for Sentiment Analysis.
- Design and build a web platform for user analysis, to define the type of analysis and the target website.
- Deepen the knowledge and usage of technologies covered in this project such as: Web Scraping, Web Servers based on Python or Polymer to create user interface.

1.3 Structure of this document

In this section we provide a brief overview of the chapters included in this document. The structure is as follows:

- ***Chapter 1*** provides an introduction to the context in which this project is developed. Besides, it describes the main objectives to achieve once concluded.
- ***Chapter 2*** offers a description of the main standards and technologies on which this project rely.
- ***Chapter 3*** details an overview of the GSI Crawler architecture.
- ***Chapter 4*** describes a selected use case.
- ***Chapter 5*** analyzes the behavior and performance of the system. In addition, the effort to build a new crawler will be measured.

- ***Chapter 6*** sums up the conclusions extracted from this project, and we offer a brief view about the lines of future work.

CHAPTER 2

Enabling Technologies

2.1 Introduction

2.2 Information retrieval

The amount of information available in the web has grown exponentially over the last years, with standards such as Linked Data helping exchange data among heterogeneous systems. However, many times these standards are not followed, and so it becomes necessary to recover and convert the data into compatible formats. There are multiple frameworks capable of crawling the web and recovering the relevant pieces of information, but we will focus here in Scrapy[1], a Python tool that allows users to extract data from websites into any format, with powerful capabilities.

2.2.1 Scrapy

Scrapy [12] is a fast high-level web crawling framework, used to extract structured data from websites. It is written in Python, and by default outputs data to JSON,

[1] http://scrapy.org/

although it accepts custom exporters giving the user the ability to export into any format it requires. Originally designed for web scraping, it can also be used to extract data using APIs or as a general purpose web crawler.

Scrapy project architecture is built around 'spiders', which are self-contained crawlers which are given a set of instructions. The spiders are Python classes that extend the Spider class in scrapy.

Listing 2.1: Amazon Scraper snippet

```
# -*- coding: utf-8 -*-
import scrapy
import re
import json
import urlparse


class AmazonScraper(scrapy.Spider):
  name = "AmazonScraper"
  allowed_domains = ["amazon.es"]
  start_urls = []
  url_reviews_format = 'http://www.amazon.es/product-reviews/%s'
  url_base = 'http://www.amazon.es/'
  pages = None
  current_page = 1

  def __init__(self, amazon_id, pages=None):
    self.start_urls = [self.url_reviews_format % amazon_id]
    if(pages == None):
      self.pages = None
    else:
      self.pages = int(pages)

  def parse(self, response):
    item = AmazonItem()
    name = response.css('div.a-row.product-title a::text')[0].extract()
    name = self.encodeUTF8(name)
    price = response.css('div.a-row.product-price-line span.a-color-price::text')[0].
        extract()
    price = self.encodeUTF8(price)

        item["url"] = response.url
        item["name"] = name
        item["price"] = price

        return item
```

Listing 2.1 shows part of a scrapy spider that will return a JSON object containing

the URL of the scrapped page, as well as the name and price fields scrapped from the document, containing an Amazon item.

2.2.2 Scrapyd

Scrapyd[2] is a service for deploying, running and managing Scrapy spiders. It's managed using a JSON API, but some online services, like Scrapy Cloud[3], have developed a web interface to make the administration easier. Scrapyd can schedule a spider run, monitorize the job and view the results through a Web interface. Finally, the job result is stored. For this purpose Scrapy provides a collection of Item Exporters for different output formats, such as XML, CSV or JSON. In addition, you can define an item pipeline to store the result in a database, using MySQL or MongoDB.

2.2.3 PhantomJS

PhantomJS[4] is a headless Webkit browser scriptable with a JavaScript API [9]. Headless browser is a web browser without a graphical user interface. Google stated in 2009 that using a headless browser could help their search engine index content from websites that use AJAX.

PhantomJS provides automated control of a web page in an environment similar to popular web browsers, but is executed via a command line interface or using network communication. It's particularly useful for testing web pages as it's able to render and understand HTML the same way a browser would, including styling elements such as page layout, color, font selection and execution of JavaScript and AJAX which are usually not available when using other testing methods.

This headless browser, PhantomJS, could be used combined with Selenium. Using both tools we can build an automated script that simulates the behavior of a normal browser. JavaScript could be rendered and, if the content is served dynamically, we could collect the information properly.

[2] http://scrapyd.readthedocs.org/
[3] http://scrapinghub.com/scrapy-cloud/
[4] http://phantomjs.org/

2.2.4 Selenium WebDriver

Selenium[5] is a web testing framework. Selenium provides the necessary tools to automate a browser giving a series of instructions and gathering the results in several ways.

Selenium WebDriver [10] is the function we are going to focus on. The WebDriver is the evolution of Selenium Remote Control. It has been developed to connect Selenium to the browser we want to use, in this case, PhantomJS. The WebDriver sends the request to the browser, the browser processes the request and sends back the result.

Using Selenium, we can code the scripts using Python like the Listing 2.2:

Listing 2.2: TripAdivsor Scraper snippet

```
from selenium import webdriver
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities
from selenium.webdriver.phantomjs.service import Service as PhantomJSService
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

def retrieveItem(url):

  item = TripAdvisorItem()

  user_agent = (
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4) " +
    "AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.57 Safari/537.36"
  )
  dcap = dict(DesiredCapabilities.PHANTOMJS)
  dcap["phantomjs.page.settings.userAgent"] = user_agent
  service_args = ['--load-images=no']
  driver = webdriver.PhantomJS(desired_capabilities=dcap,
    service_log_path=os.path.devnull, service_args=service_args)
  driver.get(url)
  name = driver.find_element_by_css_selector('#HEADING').text.encode('UTF-8')
  image = driver.find_element_by_css_selector('div.carouselPhoto.border.inView img')
  image = image.get_attribute('src')

  item["url"] = url
  item["name"] = name
  item["image"] = image

  return item
```

[5] http://docs.seleniumhq.org/

2.3 Text Analysis

2.3.1 Senpy

Senpy[6] is a tool originally developed to create sentiment and emotion analysis servers easily. It is written in Python and it uses NIF+JSON-LD [11] [16] as interface, making it suitable to communicate different NLP tools. Its goal is to provide a simple way to turn sentiment analysis algorithms into servers.

To create a web service, a new plugin must be defined. Each plugin only needs two files: a .senpy file where the service is defined, including the port where it will be accessible and other options, and a .py file where the service is implemented and the algorithm is called with the options required in the call parameters. Finally, in this Python file the response is processed, creating an "Entry" with various attributes for each element, converting them to NIF+JSON-LD and sending the final response.

2.4 Web technologies

Known simply as "The web", the World Wide Web is an information system where hypertext documents are accessed via the internet. First proposed by Tim Berners-Lee in 1989 [3], it has grown to be used by two out of five people around the world[7].

The technologies used in web services can be divided in Client technologies, executed in the user's computer, and Server technologies, executed in the server side of the service. We will provide a short description of some technologies available for each side, focusing on those used in this project.

2.4.1 Client technologies

Web browsers are usually responsible for running the code of a website. Usually, that code consists on CSS, HTML and JavaScript files, that are interpreted by the browser to present the page, and respond to the user actions.

- **HTML** or *HyperText Markup Language* is the standard markup language used to create web pages. It consists on a collection of pairs of tags that identify the

[6] https://github.com/gsi-upm/senpy
[7] http://webfoundation.org/about/vision/history-of-the-web/

different elements on a page, describing the structure of the page. It can also include images and other objects, allowing for complex user interaction.

- **CSS** or *Cascading Style Sheets* is a style sheet language, used mostly to describe the look and formatting of documents written in a markup language such as HTML. It is designed to allow separation of content from document presentation, providing more flexibility and control of the presentation, while also improving accessibility.
- **JavaScript** is a programming language used to run scripts to interact with the user inside the browser window. Along with JavaScript, it most used library[8] is jQuery, which is designed to simplify many of the usual tasks performed with JavaScript.

These technologies are often used with Ajax (*Asynchronous JavaScript XML*), a group of web development techniques used to create asynchronous web applications. Using Ajax, web applications can interact with the server in the background, therefore not interfering with the behavior and graphical display of the rest of the page.

2.4.1.1 Polymer Library

Polymer[9] is a library for creating Web Components, which are a set of W3C standards and upcoming browser APIs for defining your own custom HTML elements [13]. With the help of polyfills and sugar, it can create these custom elements and bring Web Component support to browsers that don't play nice with the standard just yet. Polymer provides a set of polyfills that enables us to use web components in non-compliant browsers with an easy-to-use framework. Polymer does this by:

- Allowing us to create Custom Elements (Listing 2.3) with user-defined naming schemes. These custom elements can then be distributed across the network and used by others with HTML Imports (Listing 2.4).
- Allowing each custom element to have its own template accompanied by styles and behavior required to use that element.
- Providing a suite of ready-made UI and non-UI elements to use and extend in your project.

[8] http://libscore.com/#libs
[9] https://www.polymer-project.org/1.0/

Polymer library has been used to build the GSI Crawler user interface.

Listing 2.3: Custom element name-tag

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

<dom-module id="name-tag">

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

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

</dom-module>
```

Listing 2.4: Usage of custom element name-tag

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

2.4.2 Server technologies

The interactions presented by the web client are the processed in the server side, usually communicating using HTTP. There are multiple applications capable of handling this interaction, known as HTTP servers. Apache, NginX or Microsoft Windows

Server® are some of the most popular servers[10].

For our service, we have used Apache [8], an Open Source HTTP-Server. First launched in 1995, it has continued development to this date, with version 2.4.17 being released on October 2015[11]. It is currently developed and maintained by an open community of developers under the Apache Software Foundation, and made available in a wide variety of operating systems, including GNU/Linux and Microsoft Windows®. It features a module based system, allowing the core functionality to be expanded by compiled modules. Some of the most popular modules include:

- **mod_php** Enabling the use PHP to execute server side code, this module can be found in many Apache installations, allowing the deployment of services like WordPress or Joomla.

- **mod_auth_basic** Handling basic user authentication, this module allows the server administrator to block sections of the server from being accessed by the general public.

- **mod_proxy** Allows the use of Apache as a proxy, masking other services behind it.

- **mod_wsgi** The aim of mod_wsgi is to implement a simple to use Apache module which can host any Python application which supports the Python WSGI interface. The module would be suitable for use in hosting high performance production web sites, as well as your average self managed personal sites running on web hosting services.

Over the last few years, there has been an important trend in the use of evented web technologies, a vision of the traditional web APIs complemented with other APIs that produce events, and provide a callback mechanism, making the Web more like a giant network. Node.js[12] is one of the most popular environments to build this kind of applications.

[10] http://news.netcraft.com/archives/2015/05/19/may-2015-web-server-survey.html
[11] http://httpd.apache.org
[12] https://nodejs.org

2.4.2.1 WSGI Servers in Python

Web Server Gateway Interface (WSGI) is a specification for simple interfaces between web servers and web applications for the Python programming language. First defined in PEP 333 [5], and updated in PEP 3333 [6], it has been adopted as a standard for Python web application development.

There are multiple implementations and frameworks of WSGI for Python, some of the most popular are:

- **Bottle** is a simple lightweight WSGI framework, focusing on simplicity [7]. It is distributed as a single-file module, with no other dependencies than the Python standard library. However, it has capabilities to handle routing, easy access to web data such as cookies and HTTP headers, and includes a built-in server for development. It also has support for templates, both with a built-in engine, and using external modules such as mako or jinja2.
- **Django** is a full-fledged Python web framework, offering fast and scalable services, with multiple built-in options, such as security and administration tools. It is slightly higher level than other frameworks, and emphasizes reusability of components and plugins, as well as rapid development.
- **Flask**, a micro web application framework, based on the jinja2 template engine, and the Werkzeug WSGI toolkit. It focuses on providing a simple interface, whilst still providing multiple features such as RESTfull request dispatching, cookies and request handling and unicode support. It also includes native support for unit testing, as well as a development server and debugger. It supports extensions for extra functionalities.

In our application, we have chosen Apache as the gateway server, using mod_wsgi, and Bottle[13] for the application itself.

[13] http://bottlepy.org/

CHAPTER 3

GSI Crawler Architecture

3.1 Introduction

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

The main purpose of this project is to provide a platform for sentiment analysis for social networks and opinions websites which users can use to analyze their own items. First, we need a scraper system to collect comments from the different websites. The scraper system is made up by spiders developed with several technologies. Yelp and Amazon spiders are developed using Scrapy framework, TripAdvisor spider is developed using Selenium+PhantomJS, and YouTube, Twitter and Facebook spiders are developed using their own API libraries. Second, we need a web application acting like an interface between the user and the scraping and analyzing system. This web application is composed of two parts: frontend and backend. The frontend is built

using Polymer library[1] and the backend is built using Bootle[2]. The backend will receive the request from frontend and it will start the crawling process and, next, it will start the analyzing process. When the analysis ends, the backend will notify to the frontend, showing the result.

Finally, a crawler schedule service have been implemented. This service is able to run crawler jobs and store the results. Later, these results can be accessed in order to be analyzed.

With all the modules above, we have a platform that allows us scraping and analyzing social networks and opinions websites comments.

A diagram of the general architecture is shown in Figure 3.1. Each module will be detailed in the following sections.

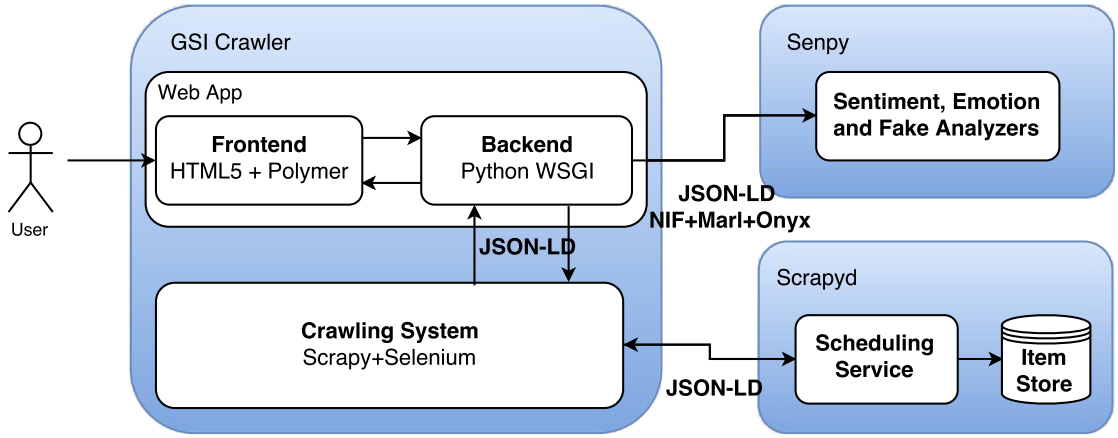

Figure 3.1: General Architecture

3.2 General Overview

The core of this project is the **GSI Crawler**. As it can be seen in Listing 3.1, GSI Crawler is composed by three different modules: *Frontend*, *Backend*, and *Crawling System*. All these modules are differentiated following functional criteria.

The interconnection of these modules into a major functionality is represented by Figure 3.1. These modules represented above are described as follows.

[1] https://www.polymer-project.org/1.0/
[2] http://bottlepy.org/

1. *Frontend*. The user interacts with the web interface to analyze items. This user interface is a web application.
2. *Backend*. The web application is supported by a backend server. The server will receive the client request and it will trigger the crawling process and, next, it will trigger the analyzing process.
3. *Crawling System*. Once an analyzing request is triggered, the first step is to retrieve the comments from the website. This system will use the specific spider to crawl the item messages the user want to analyze. When the crawling process is finished, the results are delivered back to the backend module.
4. *Scrapyd*. The service implemented is able to schedule crawling jobs to recollect messages periodically. The messages are stored and, later, they can be analyzed.

3.3 GSI Crawler

GSI Crawler is the result of connecting every module of the project. GSI Crawler has been developed as platform allocated into a server, that can be accessed using a Web Browser. This platform provides a user interface to interact with the user. The interface has been made to make easy and fast the process of analyzing and showing the results. The server side is the responsible of getting the users commands, crawl the comments and trigger the analysis. Finally, the server will send to the client the result of the analysis.

3.3.1 User Interface

The user interface is a web application. It has been developed as a clear user interface using *Material Design* principles. The interface brings a lateral menu with three sections: the functional one (*Analysis*) and two informative sections (*What is it GSI Crawler?* and *About us*). Focusing in the *Analysis* section, we can find a menu inside a floating button. In this menu, a button for each website is displayed (*Yelp*, *Amazon*, *Twitter*, *Facebook*, *YouTube* and *TripAdvisor*).

When any of these websites items are pressed, a modal window is shown. In this window, the user needs to fill the item information (like URL, Video Id, Fanpage Id or Hashtag) and the user can select what type of analysis he wants to do. This modal window is shown in Figure 3.2.

Once the analysis is finished, the result is shown in a card (Figure 3.3). The card will show the overall analysis result, but the user can review every message analysis using the *Reviews* button. A modal window will be shown with every message with the information of analysis detailed.

The user interface has been developed using Polymer framework. Polymer has been designed to make a HTML5 web as modular as possible, using its custom tags. As this project is not a closed project but it will be eventually updated, making modular could be a key factor to keep it maintainable. For this reason, Polymer has been chosen.

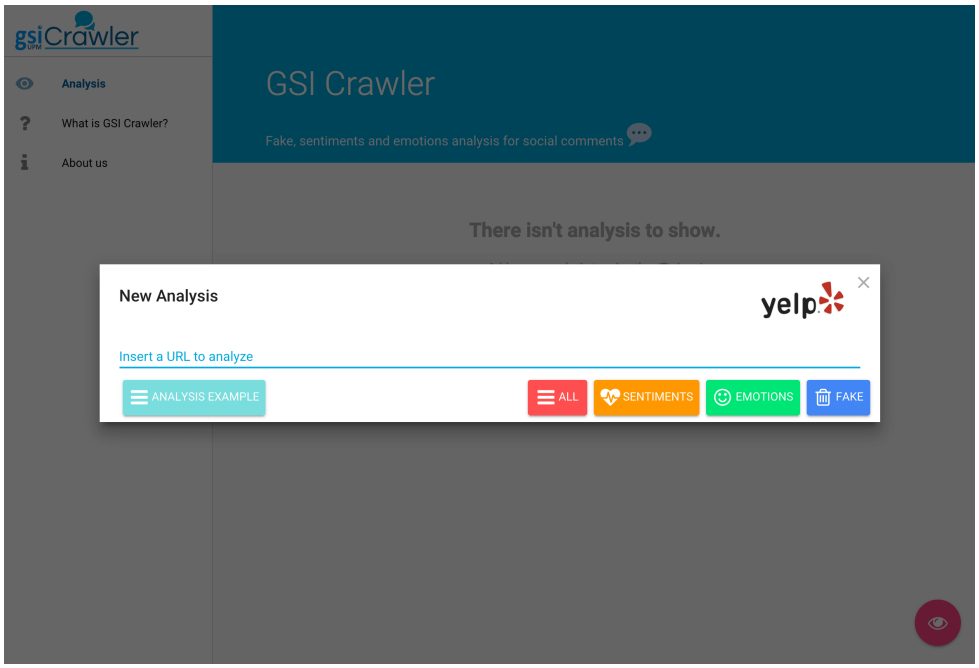

Figure 3.2: Analysis Type Modal Window - User interface

3.3.2 Server side

The web application is supported by the platform backend. For compatibility reasons, and as the entire project has been developed using Python, the server has been developed in *Python* using *Bottle*[3], a lightweight WSGI[4] micro web-framework for Python. When a user triggers an analysis using the web application, the server assigns it a job id and start the crawling process. This job id is sent back to the client, and the client must use this job id (known as UUID) to request the analysis

[3]http://bottlepy.org/
[4]https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface

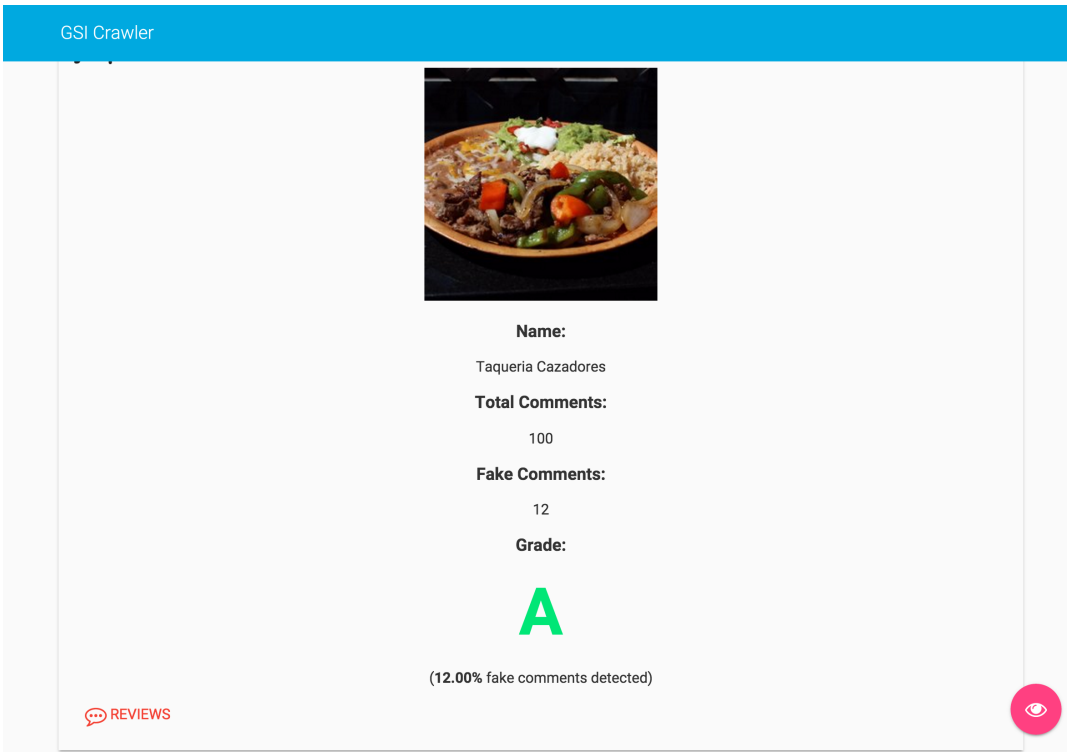

Figure 3.3: Analysis Result - User interface

result. Between modules, JSON-LD has been chosen to interchange information. JSON-LD is a lightweight Linked Data format. As JSON-LD is a Web Standard and as our system is susceptible to be connected with very different systems, to be a web standard is a key factor to be used. In addition, we have followed Schema[5] recommendations to name the different properties sent by each module. Thus, the Crawling System will store the result using a JSON-LD structure into a file with the following name *[JOB-ID].scraper*. This file is stored in a folder named *analysis*, inside web app root folder.

Listing 3.1: Crawling result structure

```
{
   "@context" : "http://schema.org",
   "name"     : "Chipotle Mexican Grill",
   "image"    : "http://s3-media1.fl.yelpcdn.com/bphoto/6LE3czYoi_YFW855C93z2w/o.jpg",
   "reviews"  : [
    {
      "@type"  : "Review"
      "author" :
      {
```

[5]https://schema.org/

```
        "@type" : "Person",
        "name"  : "Yanni L."
      },
      "datePublished" : "7/24/2015",
      "reviewBody" : "The best Chipotle location in the city!!! Great service,
          consistent quality, and generous portions!"
    },
    {
      "@type"  : "Review"
      "author" :
      {
        "@type" : "Person",
        "name"  : "Bonnie P."
      },
      "datePublished"    : "11/30/2015",
      "reviewBody" : "This location is just ok. Portion size is sooo tiny!  If I'm
          paying more than two bucks just for some guac, I better get a decent amount. "
    }
   ]
}
```

In Listing 3.1 the JSON-LD crawling result structure is shown. Other fields could be included if the website give relevant information in order to be used in future to improve the analysis.

When the crawling process is finished, the server start the analysis process. The analyzing module sends the messages to Senpy platform and get the analysis result for every message. Communication between Web App and Senpy platform is established using JSON NIF+Marl [2] (Sentiments Analysis) and JSON NIF+Onyx [15] (Emotion Analysis). Once every message is analyzed, the analysis module gets the overall analysis result and store it using a JSON structure in a file with the following name *[JOB-ID].analysis*. This file is stored in a folder named *analysis*, inside web app root folder.

Listing 3.2: Analysis result structure

```
{
   "@context" : "http://schema.org",
   "name"     : "Chipotle Mexican Grill",
   "image"    : "http://s3-media1.fl.yelpcdn.com/bphoto/6LE3czYoi_YFW855C93z2w/o.jpg",
   "reviews"  :
     [
    {
      "@type"  : "Review",
      "author" :
      {
```

```
      "@type" : "Person",
      "name"  : "Yanni L."
    },
    "datePublished" : "7/24/2015",
    "reviewBody" : "The best Chipotle location in the city!!! Great service,
        consistent quality, and generous portions!",
          "sentimentAnalysis" : {
              "@id": "_:b0",
              "analysis": [
                  {
                      "version": "0.1",
                      "@id": "sentiment140_0.1",
                      "maxPolarityValue": 1.0,
                      "name": "sentiment140",
                      "minPolarityValue": 0.0
                  }
              ],
              "entries": [
                  {
                      "text": "The best Chipotle location in the city!!! Great service
                          , consistent quality, and generous portions!",
                      "@id": "Entry0",
                      "nif:language": "auto",
                      "opinions": [
                          {
                              "marl:hasPolarityValue": 1.0,
                              "prov:wasGeneratedBy": "sentiment140_0.1",
                              "@id": "Opinion0",
                              "marl:hasPolarity": "marl:Neutral"
                          }
                      ]
                  }
              ]
    }
  },
  {
    "@type"  : "Review",
    "author" :
    {
      "@type" : "Person",
      "name"  : "Bonnie P."
    },
    "datePublished"    : "11/30/2015",
    "reviewBody" : "This location is just ok. Portion size is sooo tiny!  If I'm
        paying more than two bucks just for some guac, I better get a decent amount.
        ",
          "sentimentAnalysis" : {
              "@id": "_:b0",
              "analysis": [
                  {
                      "version": "0.1",
                      "@id": "sentiment140_0.1",
```

```
                        "maxPolarityValue": 1.0,
                        "name": "sentiment140",
                        "minPolarityValue": 0.0
                    }
                ],
                "entries": [
                    {
                        "text": "This location is just ok. Portion size is sooo tiny!
                            If I'm paying more than two bucks just for some guac, I
                            better get a decent amount.",
                        "@id": "Entry0",
                        "nif:language": "auto",
                        "opinions": [
                            {
                                "marl:hasPolarityValue": 0.5,
                                "prov:wasGeneratedBy": "sentiment140_0.1",
                                "@id": "Opinion0",
                                "marl:hasPolarity": "marl:Neutral"
                            }
                        ]
                    }
                ]
      }
    }
   ],

   "loading" : false,
   "error"   : null
}
```

In Listing 3.2 the JSON analysis result structure is shown. Other fields could be included if the website give relevant information.

This JSON is delivered to client when the analyzing process is finished. If the client request the analysis result and the result is not ready, the following JSON will be delivered (Listing 3.3).

Listing 3.3: Analysis not ready structure

```
{
   "error"    : null,
   "loading"  : false,
   "id"       : "f711d419-00f8-495b-bd6f-7d9774f14181"
}
```

If an error occurs, in both crawling process and analyzing result steps, the last JSON structure will be used too. The *error* field must have any value except *null*.

loading field will be always false in case of error. *id* field gives the analysis id.

3.3.3 Crawling System

The Crawling System is the most important module inside GSI Crawler. This module is the responsible to recollect the messages from the different websites. According to the specific needs, for each website a crawler has been developed using different technologies.

The development of each crawler will be detailed in the next lines, explaining the difficulties found in the development process.

3.3.3.1 Amazon

The Amazon Crawler has been developed using Scrapy, written in Python. Scrapy is one of the fastest and easiest of implement crawlers. In addition, Scrapy is fully scalable giving a good support to run parallels crawling jobs at the same time. For this reason, Scrapy is the best option to be used with Amazon. Amazon lets users leave reviews about the items that they have bought. Every item has a unique id, a ten character alphanumeric string. Amazon offers a REST structure for their website. Product reviews can be found following the next REST structure: *http://www.amazon.es/product-reviews/[AMAZON-ID]*. At this URL, the crawler can find relevant information like the item's title, the item's price, the overall rating and a list of reviews. Each review is shown with the next relevant information: author's name, rating, date and message. The list of reviews is divided using pagination and each page could be accessed using a simple URL link. Once the crawler gets the HTML code of the website, the information is gathered using CSS Selectors. As Amazon doesn't use a standard to tag the HTML fields, this crawler could fail if Amazon does some modifications in the user interface of their web page.

An example of the JSON generated by the Amazon Crawler is shown at Listing A.2

3.3.3.2 Yelp

The Yelp Crawler has been developed using Scrapy, written in Python. Yelp is a platform where the users leave comments about several kinds of business they have

visited. Inside Yelp, each business has a unique name, and this business can be accessed following the next URL: *http://www.yelp.com/biz/[BUSINESS-NAME].* At this URL, the crawler can find relevant information like the business's name, the business's price range, the overall rating and a list of reviews. Each review is shown with the next relevant information: author's name, rating, date and message. The list of reviews are divided using pagination and each page could be accessed using a simple URL link. Once the crawler get the HTML code of the website, the information is gathered using CSS Selectors. Yelp uses Schema[6] standard to tag their relevant information. That make the job of gathering the information easier. In addition, the crawler will work even if Yelp does important modification of their user interface of their web page if they continue using Schema standard.

An example of the JSON generated by the Yelp Crawler is shown at Listing A.1

3.3.3.3 TripAdvisor

TripAdvisor is a platform like Yelp, but it is focused on travel business. Several kinds of businesses can be found in TripAdvisor like Hotels and Restaurants. Inside TripAdvisor, each business has a unique name, and this business can be accessed following the next URL: *http://www.tripadvisor.com/[BUSINESS-NAME].* At this URL, the crawler can find relevant information like the business's name, the business's price range, the overall rating and a list of reviews. Each review is shown with the next relevant information: author's name, rating, date and message. The list of reviews are divided using pagination. Each page needs to be loaded running a script written in Javascript. Scrapy is not able to execute Javascript code. For this reason, the TripAdvisor Crawler has been written in Python using Selenium package and PhantomJS Browser. PhantomJS Browser is used to load and render the final HTML code and Selenium is used to control the browser from Python script. Once the crawler get the final HTML code of the website, the information is gathered using CSS Selectors. As TripAdvisor doesn't use a standard to tag the HTML fields, this crawler could fail if TripAdvisor does some modifications in the user interface of their web page.

An example of the JSON generated by the TripAdvisor Crawler is shown at Listing A.6

[6] https://www.schema.org/

3.3.3.4 Twitter

Twitter is a social microblogging section. Each Twitter user has his own timeline, a list of their blog entries (known as *tweets*). Twitter offers an API to access to the information needed for this project: recollect the *tweets* that contain a specific hashtag. Although the use of Scrapy is an option, the API is considerably much faster. The Twitter Crawler has been developed using Python and Twython[7]. Twython is a Python library providing an easy way to access Twitter data, a wrapper for the Twitter API. To use Twitter API, a Twitter App Key must be provided. Using Twitter API, last tweets can be filtered by a hashtag. Tweets can be retrieved with the next relevant information: author's name, date and message.

An example of the JSON generated by the Twitter Crawler is shown at Listing A.3

3.3.3.5 YouTube

YouTube is a social video-sharing network. Each video can be commented by YouTube users. YouTube offers an API to access to the information needed: recollect the comments from a specific video. Although the use of Scrapy is an option, the API is considerably much faster. The YouTube Crawler has been developed using Python and Google API Python Client library. Google API Python Client is the Python client library for Google's discovery based APIs made by Google. To use Google API, the *client_id* and *client_secret* must be provided. Using Google API, comments of a YouTube video can be gathered. These comments can be retrieved with the next relevant information: author's name, date and message.

An example of the JSON generated by the YouTube Crawler is shown at Listing A.5

3.3.3.6 Facebook

Facebook is a social network which lets the businesses create their own fan page. In the fan page, the businesses can write posts, like a blog entry. In each entry, the users can leave their comments. GSI Crawler is focused on analyzing the users comments of the blog entries from Facebook fan pages. Facebook offers an API, called *Graph API*. The Facebook Crawler is able to retrieve the comments from a fan page. It is

[7] https://github.com/ryanmcgrath/twython

written in Python and uses Graph API. To use Graph API, an OAuth Token must be provided. Using Graph API, comments from a fan page can be gathered. These comments can be retrieved with the next relevant information: author's name, date and message.

An example of the JSON generated by the Facebook Crawler is shown at Listing A.4

3.4 Crawler scheduler

The Crawler Scheduler is a service that is able to schedule the execution of the crawlers periodically. This service has been deployed using Scrapyd which is able to manage natively Scrapy crawlers. In addition, using a web interface (Figure 3.4), the execution of the crawlers can be monitored and the results of the crawling process can be viewed. This service can be managed using a JSON API. The crawling result can be retrieved anytime using this JSON API.

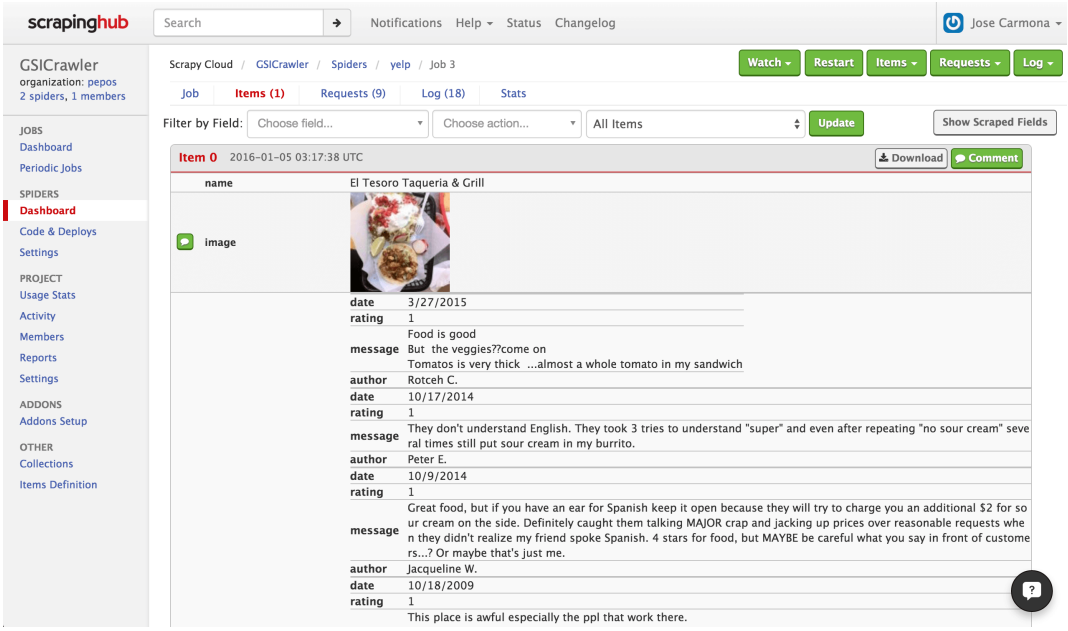

Figure 3.4: Scrapy Cloud Web Interface

CHAPTER 4

Case study

4.1 Introduction

In this chapter we are going to describe a selected use case. This description will cover the main GSI Crawler features, and its main purpose is to completely understand the functionalities of GSI Crawler, and how to use it.

The actor of this case is the user. The goal of the user is to perform a brand monitoring, choosing between 6 different websites and choosing between 3 kinds of analysis. In this use case the user wants to monitor and analyze the comments from different websites. For instance, if the user wants to monitor a restaurant in different social networks, such as: Yelp, Twitter, Facebook, YouTube and TripAdvisor.

Finally, we are going to explain how to schedule a new crawler job, storing the results. These results could be retrieved anytime the user wants.

4.2 Performing a new analysis

The first step to take is to browse to GSI Crawler main page (Figure 4.1). In the main page, at the Analysis lateral menu option, a menu containing the different websites options can be found, inside a floating button with an eye icon.

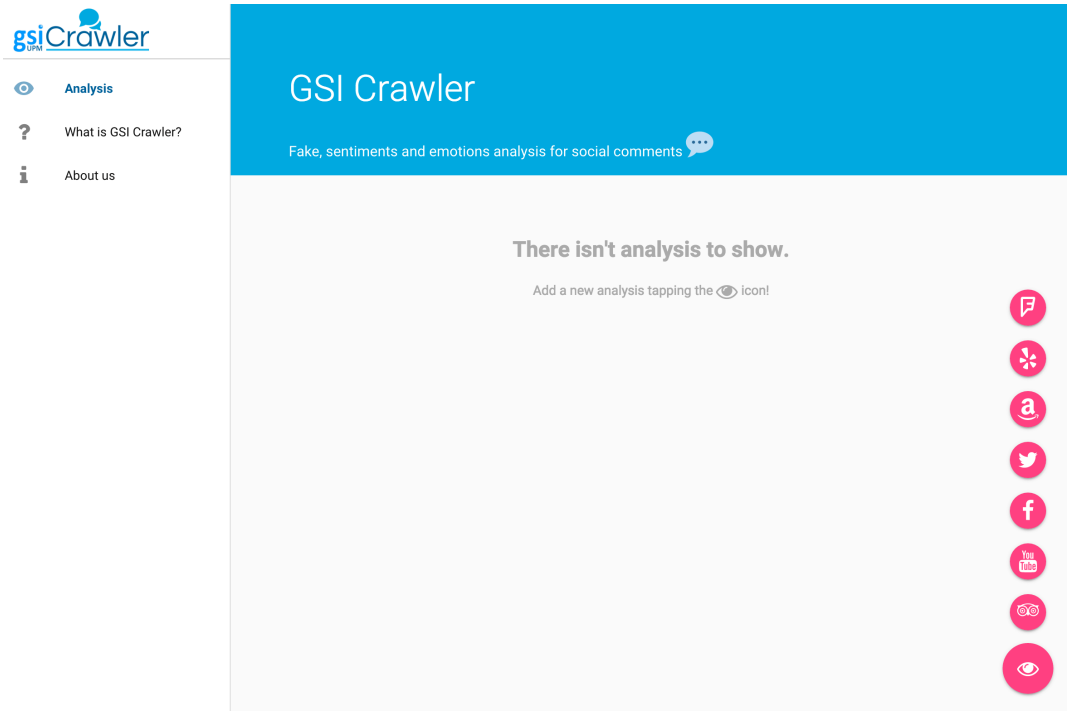

Figure 4.1: Main Page

In this menu, the website where the comments will be extracted can be chosen. To illustrate this use case in detail, a restaurant with presence in Yelp, Twitter, Facebook, YouTube and TripdAvisor will be used. Choosing an option from this menu, a modal window (Figure 4.2) will be opened.

This window has a field to introduce the URL from where the comments will be extracted. This URL points to a item page in case of Amazon or to a business page in case of Yelp or TripAdvisor. If Twitter, Facebook, or YouTube option is chosen, the user has to introduce in this field a Twitter hashtag, a fan page Facebook id or a YouTube video id, respectively.

At the bottom of the new analysis modal window, four different buttons can be find. The *Fake* button will start a fake analysis, the *Emotions* button will start an emotion analysis, *Sentiments* button will start a sentiment analysis, and the *All*

button will start the three previous analysis at the same time. Furthermore, a button named "Analysis Example" can be found. Tapping on this button, an example URL value will fill the URL field.

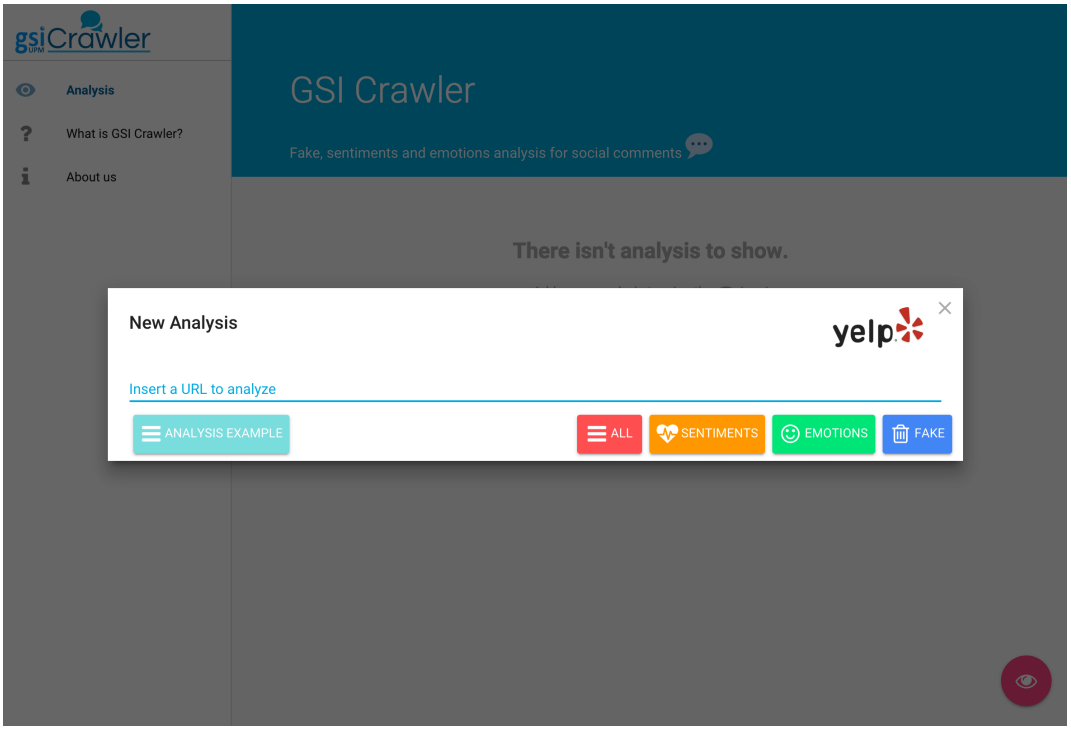

Figure 4.2: New analysis modal window

4.2.1 Fake Analysis

This kind of analysis will mark those comments that are not real, called fake comments. These false comments are made to increase (or decrease) the business social reputation, trying to cheat the websites rating system.

To make a fake analysis, the user needs to choose one option between the different websites, fill the URL field and press *Fake* button. A loading card (Figure 4.3) will be added to the main page and it will display the result when the server ends the analysis (Figure 4.4).

The card will show the overall analysis result taking in account every comment. We can see how many comments in total are scraped and analyzed and how many comments are marked as *Fake* comments. Based on the percentage of fake comments over the total number, a grade is displayed with an associated color.

Finally, pressing *Reviews* button a list of every comment scraped and analyzed is displayed (Figure 4.5). In this list we can find what comments are marked as *Fake* comments, its author, its date, its rating value, and its message.

An example of the JSON generated by the Fake Analyzer is shown at Listing A.1

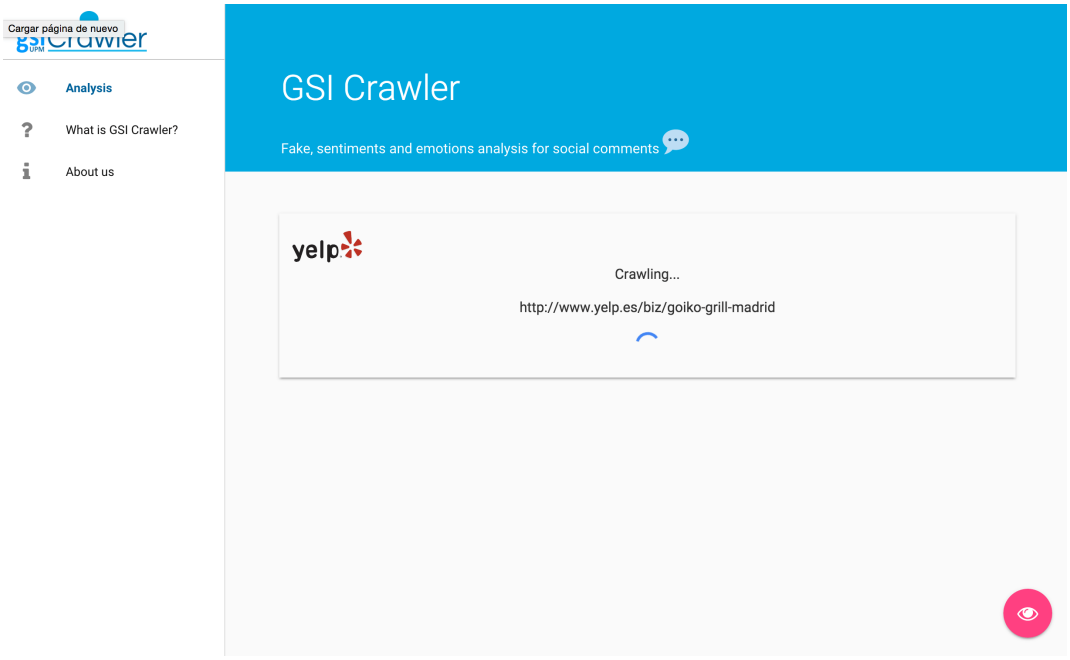

Figure 4.3: Loading card

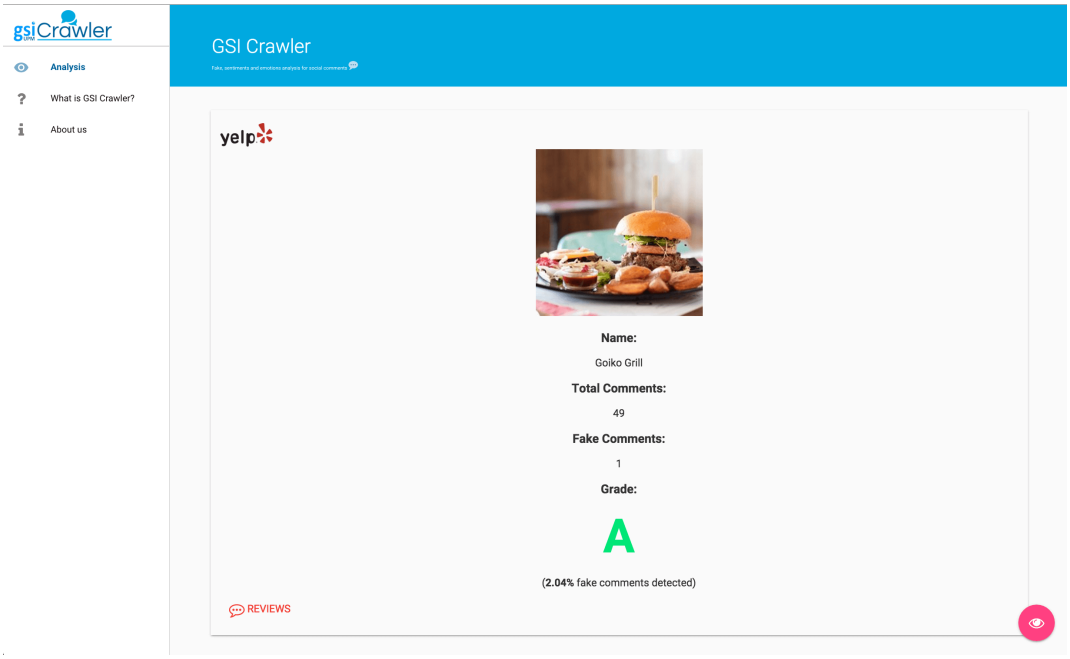

Figure 4.4: Fake Analysis Result

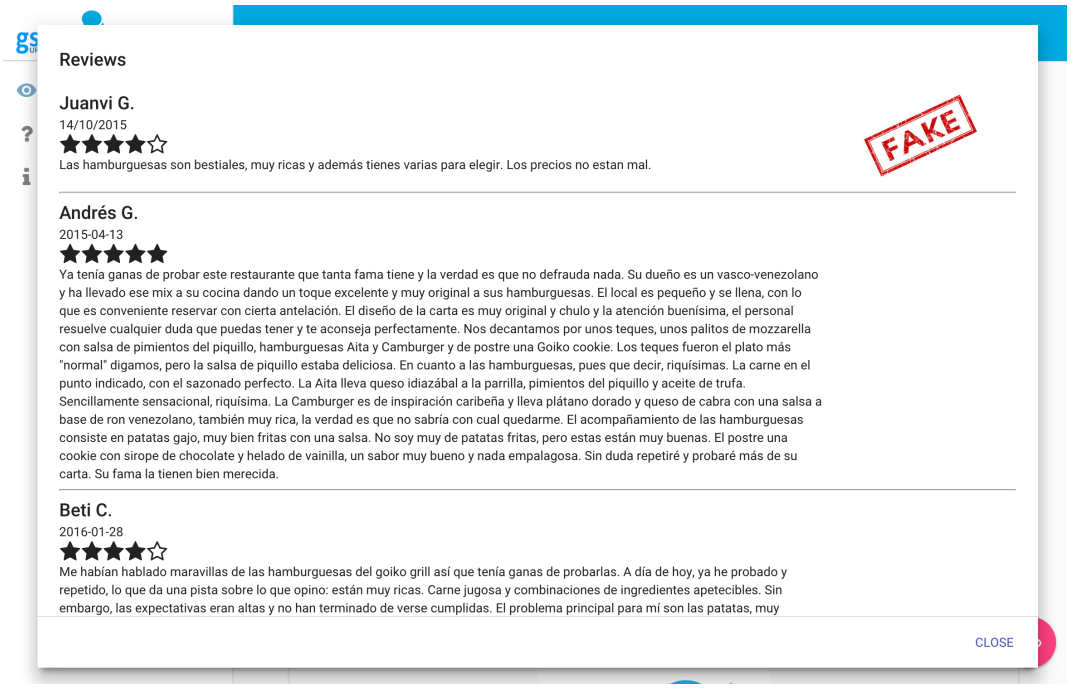

Figure 4.5: Fake analysis result detailed each comment

4.2.2 Emotions Sentiments

This kind of analysis will rate the scraped comments based on the emotions extracted from the message. The analyzer will rate these messages with different emotions: *disgusted*, *angry*, *afraid*, *sad*, *happy*, and *surprised*. These emotions will be associated with a specific color to show the results.

To make an emotion analysis, the user needs to choose one option between the different websites, fill the URL field and press *Emotions* button. A card (Figure 4.3) will be added to the main page and it will display the result when the server ends the analysis (Figure 4.6).

The card will show the overall analysis result taking in account every message. Based on the emotions of all comments, an overall emotion rate is displayed with an associated color and emoticon.

Finally, pressing *Reviews* button a list of every comment scraped and analyzed is displayed (Figure 4.7). In this list we can find what emotion is predominant in each comment, its author, its date, and its message.

An example of the JSON generated by the Emotion Analyzer is shown at Listing A.2

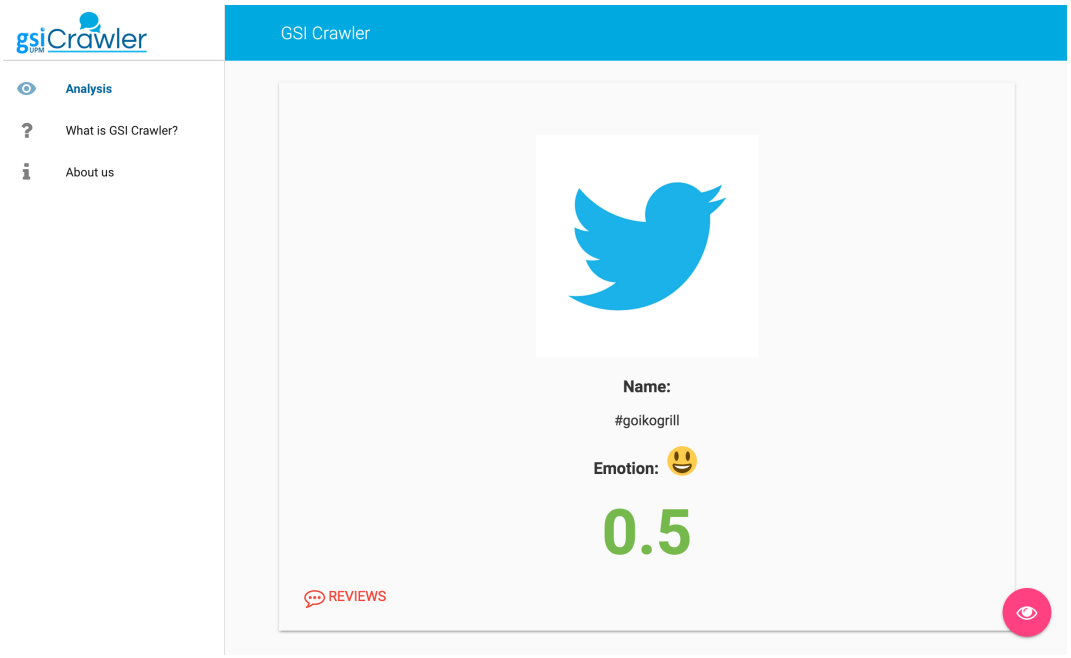

Figure 4.6: Emotion analysis result

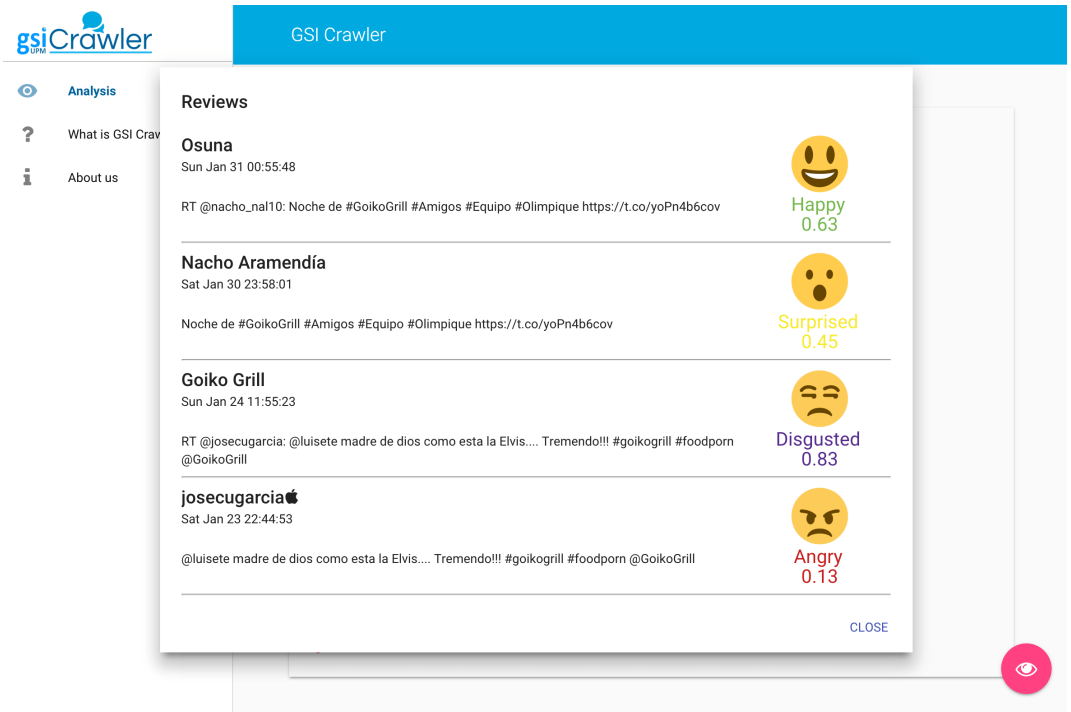

Figure 4.7: Emotion analysis result detailed each comment

4.2.3 Sentiment Analysis

This kind of analysis will rate the scraped comments based on the sentiments extracted from the message. The analyzer will rate these comments with different sentiments: *Positive*, *Neutral*, and *Negative*. These sentiments will be associated with a specific color to show the results.

To make an emotion analysis, the user needs to choose one option between the different websites, fill the URL field and press *Sentiments* button. A card (Figure 4.3) will be added to the main page and it will display the result when the server ends the analysis (Figure 4.8).

The card will show the overall analysis result taking in account every comment. Based on the sentiment of all comments, an overall sentiment polarity is displayed with an associated color.

Finally, pressing *Reviews* button a list of every comment scraped and analyzed is displayed (Figure 4.9). In this list we can find what sentiment is predominant in each comment, its author, its date, and its message.

An example of the JSON generated by the Sentiment Analyzer is shown at Listing A.3

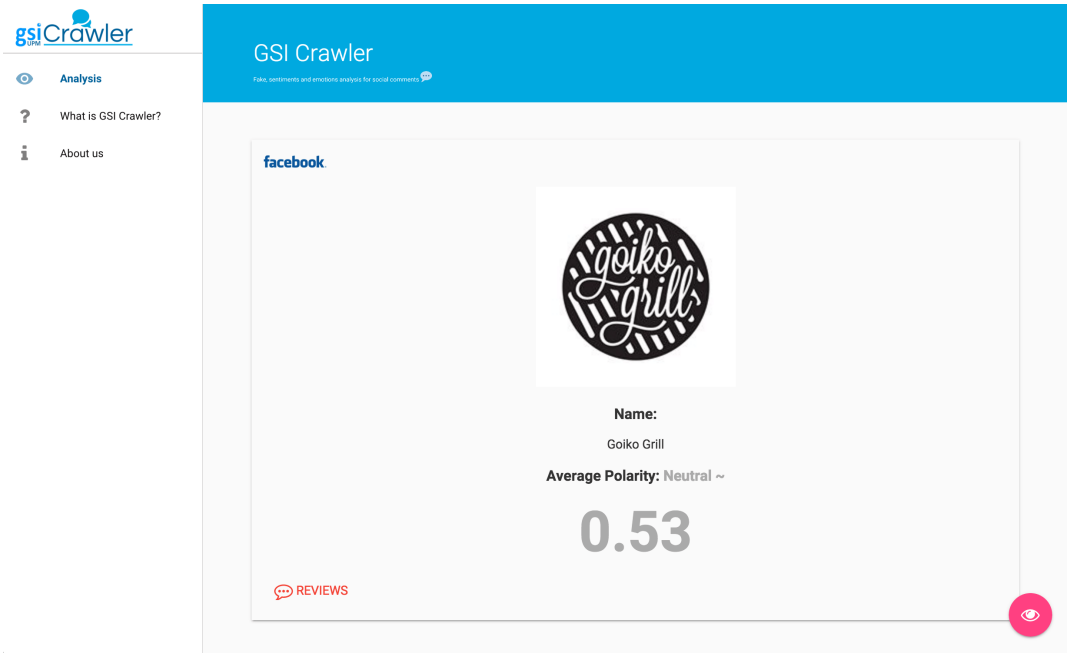

Figure 4.8: Sentiment analysis result

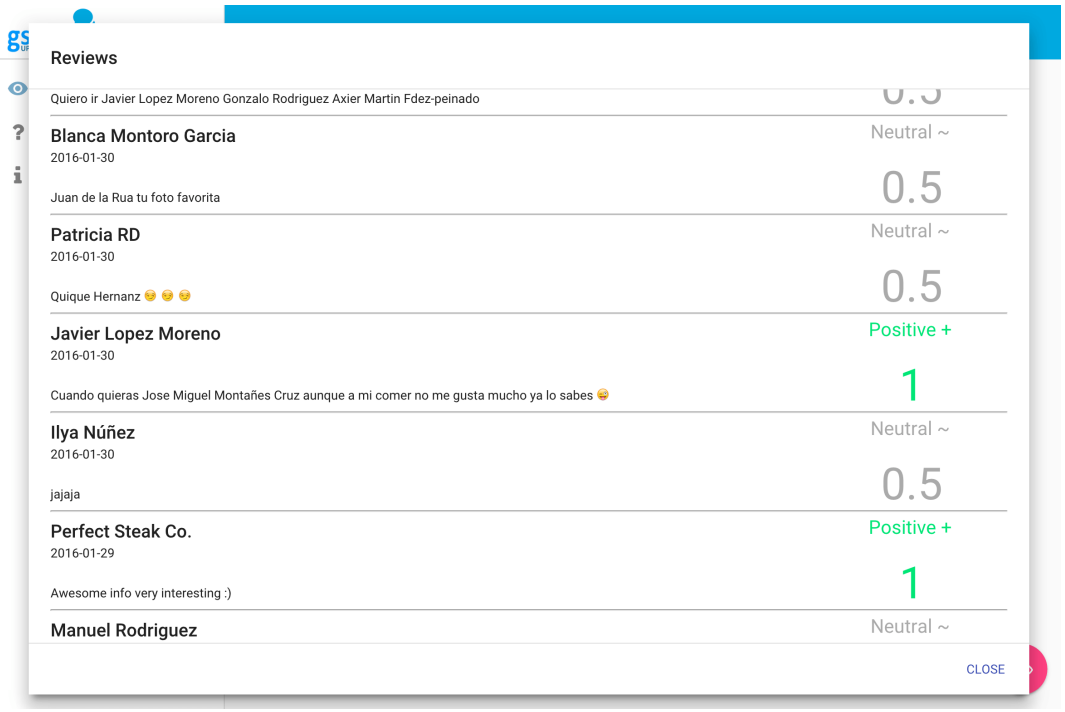

Figure 4.9: Sentiment analysis result detailed each comment

4.2.4 Perform all analysis at the same time

If the user wants to analyze fake comments, its emotions and sentiments, he could use *All* button. This will scrap comments and it will make the three different analysis, shown in a card (Figure 4.10).

An example of the JSON generated by the multipler analyzer is shown at Listing A.4

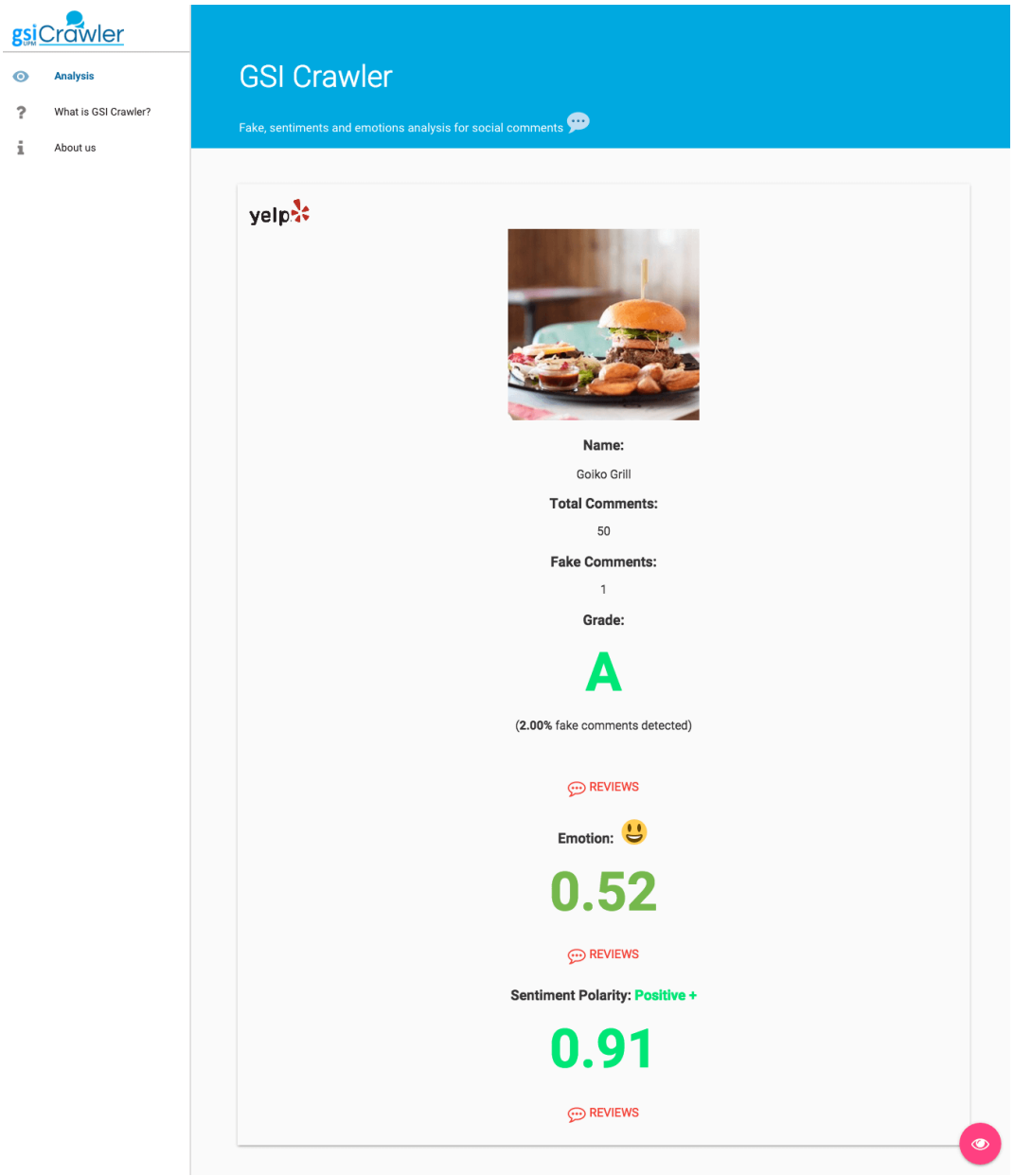

Figure 4.10: All analysis result cards

4.3 Scheduling a crawling job

To schedule a crawling job, the user could choose between 2 options: do a JSON call (using *scrapyd*[1] API, for instance, performing a curl request using the command line) or use a **Web Interface**.

Going to scraping web platform, the user could schedule a new crawling job pressing *Run Spider* button. In the modal form, the user needs to choose the spider to use (the website where the comments the user wants to analyze are) and an argument called *url* should be specified (Figure 4.11).

Once *scrapyd* has finished the crawling job, the user could get the comments scraped looking for the job in the *Completed Jobs* box and pressing on the number of items (Figure 4.12).

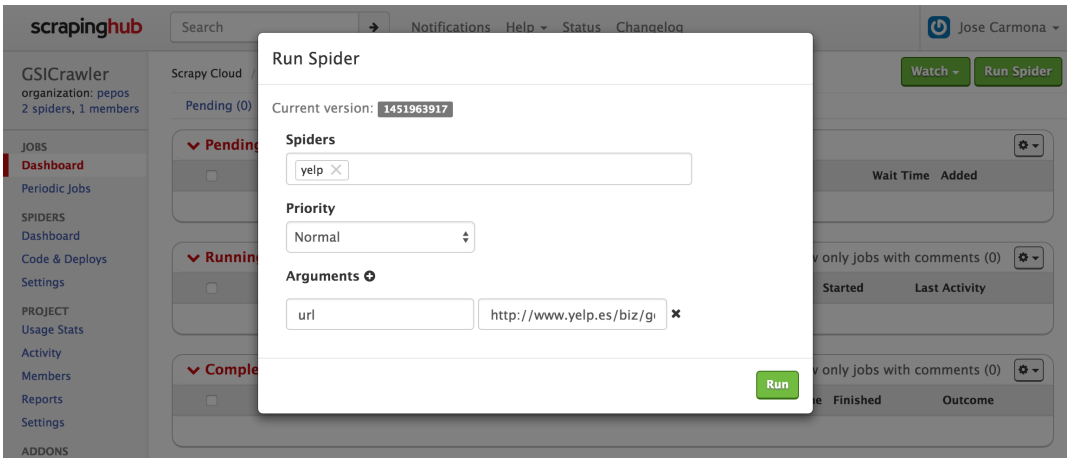

Figure 4.11: Schedule a new crawling job

[1] https://scrapyd.readthedocs.org/en/latest/

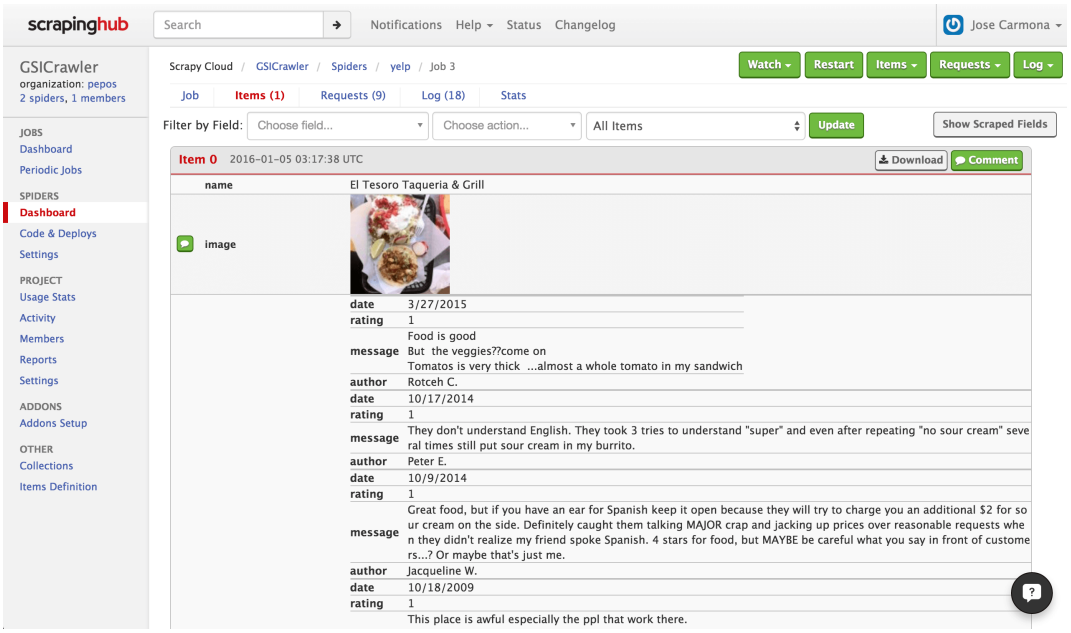

Figure 4.12: Scraped Result

CHAPTER 5

Evaluation

5.1 Introduction

In this chapter we will analyze the behavior and performance of the system. We will evaluate the time the crawler system need to extract comments from the different websites. The crawling system developed for this project has multiple heterogeneous crawlers, each one of them with different hardware requirements. Therefore, we will take a look at the performance for each crawler.

In addition, we will also evaluate the effort to write a new crawler embedded in the current system. The time and the average of numbers of lines required to build a new crawler will be analyzed.

5.2 Requirements and Benchmark

For our crawling system, we will analyze the memory and CPU usage when the different crawlers are launched to recollect the comments. As each crawler is different from the others, we have made a benchmark for each one. The software and hardware

used to obtain the benchmark results are shown in Table 5.1. The obtained results can be seen in Table 5.2.

Operating System	Ubuntu 14.04 x64, Kernel Linux 3.19.0
CPU	Intel Core i7-4750HQ 2.0GHz
Memory	2 x Hynix 4GB 1600 MHz DDR3
Hard Drive	SanDisk SSD SM0256F 256GB

Table 5.1: Software and Hardware used to benchmark the crawling system

Crawler	CPU Usage	Memory	Time to crawl 100 comments
Yelp	0.5 %	48.8 MB	9”
Amazon	0.7 %	44.3 MB	13”
Twitter	0.1 %	23.2 MB	2”
Facebook	0.1 %	20.5 MB	5”
Youtube	0.1 %	18.7 MB	3”
TripAdvisor	32 %	405.5 MB	49”

Table 5.2: Memory and CPU usage for each crawler

We can see that Facebook, YouTube and Twitter are faster than others to crawl the comments. The reason is simple, the crawling system is using their own JSON API to recollect the comments. In addition, a notable difference between Yelp and Amazon in front of TripAdvisor can be observed. The reason of this difference is while Yelp and Amazon uses Scrapy framework, TripAdvisor uses a browser (PhantomJS). The browser needs to render each visited link, and that makes the crawling process much slower and a high consumer of memory and CPU.

5.3 Effort to build a new crawler

To analyze the effort to build a new crawler, we have selected the FourSquare[1] web page. This site is very similar to Yelp and doesn't need to render Javascript, so we will use Scrapy framework to build this crawler and analyze the effort to implement to our crawling system.

We can start the development of our new crawler using a basic prepared template like Figure 5.1.

Listing 5.1: Crawling basic structure template

```
# -*- coding: utf-8 -*-
import scrapy

class FourSquareSpider(scrapy.Spider):
  name = "FourSquareScraper"
  allowed_domains = ["foursquare.es", "foursquare.com"]
  start_urls = []
  current_page = 1
  filePath=None
  handle_httpstatus_list = [404]

  def __init__(self, url, filePath=None):
    self.start_urls = [url]
    self.filePath = filePath

  def parse(self, response):
    pass

  def get_next_request(self, response):
    pass

  def parse_reviews_list(self, response):
    pass

  def extract_reviews_list(self, response):
    pass

  def parse_review_code(self, review_code):
    pass

  def encodeUTF8(self, string):
    if isinstance(string, unicode):
      return string.encode('utf-8')
    return string
```

[1] http://www.foursquare.com

```
def saveJSON(self, json, fileName):
  textutf8 = json.encode('UTF-8')
  text_file = open(fileName, "w")
  text_file.write(textutf8)
  text_file.close()
  return

def returnError(self, error):
  item = {'error':'Crawler has failed to fetch the comments', 'loading':False}
  self.saveJSON(json.dumps(item), self.filePath)
```

Helping us with web developer tools like those Chrome includes, we can get the *CSS Selectors* pointing to information we need to recollect. With few coding, we could fill empty functions and we could get a functional crawler to recollect FourSquare reviews. We can see the result in Figure 5.2.

Listing 5.2: FourSquare Crawler

```
def parse(self, response):
  try:
    name = response.css('#container > div > div.contents > div.wideColumn > div.
        venueInfoSection > div.venueHeader > div.primaryInfo > div.venueNameSection >
        h1::text')[0].extract()
    name = self.encodeUTF8(name)
    image = response.css('#container > div > div.contents > div.wideColumn > div.
        venueInfoSection > div.venueHeader > div.mainIconWrapper > a > img::attr(src)
        ')[0].extract()
    image = self.encodeUTF8(image)
    reviews = self.extract_reviews_list(response)
    item = {'name':name, 'image':image, 'reviews':reviews}
    self.current_page = 1
    nextRequest = self.get_next_request(response)
    if(nextRequest == None):
      self.saveJSON(json.dumps(item), self.filePath)
      yield item
      return
    nextRequest.meta['item'] = item
    yield nextRequest
  except Exception as e:
    self.returnError(e)

def get_next_request(self, response):
  try:
    if(self.pages != None and self.current_page >= self.pages):
      return None
    pagination_div = response.css('#tipsContainer > div.tipPagination')
    if(len(pagination_div) == 0):
```

```
      return None
    pagination_div = pagination_div[0]
    number_of_pages = pagination_div.css('a')
    if(len(number_of_pages) <= self.current_page):
      return None
    self.current_page = self.current_page + 1
    url = '%s?tipsPage=%s' % (self.start_urls[0], self.current_page)
    print url
    return scrapy.Request(url, callback=self.parse_reviews_list)
  except Exception as e:
    print str(e)
    return None
  return None

def parse_reviews_list(self, response):
  reviews = self.extract_reviews_list(response)
  item = response.meta['item']
  for review in reviews:
    item['reviews'].append(review)
  self.current_page = self.current_page + 1
  nextRequest = self.get_next_request(response)
  if(nextRequest == None):
    self.saveJSON(json.dumps(item), self.filePath)
    yield item
    return
  nextRequest.meta['item'] = item
  yield nextRequest

def extract_reviews_list(self, response):
  try:
    reviews = list()
    reviews_list_container = response.css('#tipsList')[0]
    reviews_list = reviews_list_container.css('li')
    for review_code in reviews_list:
      review = self.parse_review_code(review_code)
      if(review != None):
        reviews.append(review)
    return reviews
  except Exception as e:
    self.returnError(e)

def parse_review_code(self, review_code):
  try:
    author = review_code.css('div.tipInfo > span.userName > a::text')[0].extract()
    author = self.encodeUTF8(author)
    date = review_code.css('div.tipInfo > span.tipDate::text')[0].extract()
    date = self.encodeUTF8(date)
    message = review_code.css('p.tipText::text')[0].extract()
    message = self.encodeUTF8(message)
    return {'author':author, 'date':date, 'message':message}
  except:
    return None
```

That is, we have a 100% functional crawler in only 115 lines. Comparing with the number of lines contained in the template, we have added a total of 71 lines. Measuring the effort in time terms, this example has been coded in approximately 1 hour. It's clear that the effort to code a new crawler is very light.

CHAPTER 6

Conclusions and future work

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

6.1 Conclusions

We started this document specifying several requirements and goals we aimed to achieve with the architecture developed for this Final Project.

In this project we have created a crawling system in order to automatize the gathering of social comments from several websites.

We have used this crawling system to build a platform that allow us to crawl and analyze comments from several websites using a graphical interface, making this job easier. Senpy platform has been used to analyze the messages.

In addition, this project allows the user to schedule the crawling process to retrieve the comments from a website periodically. The conclusions deduced from the work will now be presented.

Finally, doing this work the knowledge has been deepened about the usage of technologies used in this project such as: Web Scraping, Web Servers based on python or Polymer to create user interface.

6.2 Achieved goals

In chapter 1 we discussed the goals we wanted to achieve with this project. This can be summarize as follows:

1. **Design and build a scraper for each social network (Twitter, Facebook, YouTube), for each opinions website (Yelp, TripAdvisor) and for Amazon**. One of the main requirements of this project is to develop a crawling system that automatize the gathering of social comments from several websites. The crawling module has been the core of the GSI Crawler platform developed in this project.

2. **Build a software module which is able to communicate with the Senpy platform for Sentiment Analysis**. A module to analyze the messages gathered by the crawling system has been implemented by the server built in this project. This module uses Senpy to analyze the messages.

3. **Design and build a web platform for user analysis, allowing users define the type of analysis and the target website**. The result of this goal is the platform GSI Crawler. This platform lets the user select the target website and the type of analysis and, finally, perform the analysis.

4. **Deepen the knowledge and usage of technologies covered in this project such as: Web Scraping, Web Servers based on Python or Polymer to create user interface**. The goal is clearly achieved. I never used some technologies needed to complete this project and now I acquired some ease in the usage of these technologies.

6.3 Future Work

There are several lines that can be followed to extend some used features on the sentiment analysis but were not included into this project due to the time limitation.

In the next points some lines of work or improvement to continue the development are presented.

- Extend GSI Crawler, adding user authentication and giving the option to store the different analysis.
- Build a panel to show and analyze the Crawler Scheduler results. This will be useful to monitor the evolution of reviews of a specific business or the sentiments of the public about a specific topic.
- Improve TripAdvisor crawler. Using a browser to render the Javascript code makes the process much slower. An alternative like ScrapyJS[1] could be used to build this crawler.
- Store the information using a database engine like MongoDB instead of store using JSON in a plain document.
- Use Fake analysis previously before make an Emotion or a Sentiment analysis. Discarding fake comments will increase the reliability of the final analysis.

[1] https://github.com/scrapinghub/scrapy-splash

Bibliography

[1] Web scraping. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, page 3490. Springer US, 2009.

[2] Adam Adam Westerski and J. Fernando Sánchez-Rada. Marl ontology specification. 2013.

[3] Tim Berners-Lee. Information management: A proposal. Technical report, 1989.

[4] Francesco Bonchi, Carlos Castillo, Aristides Gionis, and Alejandro Jaimes. Social network analysis and mining for business applications. *ACM Trans. Intell. Syst. Technol.*, 2(3):22:1–22:37, May 2011.

[5] P. J. Eby. Python web server gateway interface, 2003. Tech. Rep. PEP 333.

[6] P. J. Eby. Python web server gateway interface, 2010. Tech. Rep. PEP 3333.

[7] Justin Ellingwood. How to use the Bottle Micro Framework to Develop Python Web Apps. December 2013.

[8] Roy T. Fielding and Gail E. Kaiser. The Apache HTTP Server Project. *IEEE Internet Computing*, 1(4):88–90, 1997.

[9] Klint Finley. PhantomJS: The Power of WebKit but Without the Broswer. March 2011.

[10] Ayush Gondhali, Rishikesh Chandra, Ashish Shinde, and Sanket Pimple. Automation testing using data driven approach. *International Journal on Recent and Innovation Trends in Computing and Communication*, 3(4):1841–1844, april 2015.

[11] Sebastian Hellmann, Jens Lehmann, Sören Auer, and Martin Brümmer. Integrating nlp using linked data. In *12th International Semantic Web Conference, 21-25 October 2013, Sydney, Australia*, 2013.

[12] Newcoder. Introduction to Web Scraping using Scrapy. July 2013. `http://newcoder.io/scrape/intro/`.

[13] Jarrod Overson and Jason Strimpel. *Developing Web Components - UI from jQuery to Polymer.* O'Reilly, 2015.

[14] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1–135, 2008.

[15] J. Fernando Sánchez-Rada and Carlos A. Iglesias. Onyx: A Linked Data Approach to Emotion Representation. *Information Processing & Management*, 52:99–114, January 2016.

[16] Manu Sporny, Gregg Kellogg, and Markus Lanthaler. JSON-LD Syntax 1.0. Technical report, 12 2012.

APPENDIX A

Analysis Results

This chapter shows the results given by the scraping and analyzing modules for each website. The results have been trimmed (only the number of comments) in order not to extend this section.

A.1 Yelp

The crawling system recollect some information from a Yelp business. It takes the name of the business (tagged as *name*) and the URL to the main image of the business (tagged as *image*). From each comment the information recollected is: date of publication (tagged as *datePublished*), the user rating (tagged as *ratingValue*), the review message (tagged as *reviewBody*) and the author, containing its name. This result is from a Fake Analysis, the analysis is under *fakeAnalysis* key. It contains a Boolean property (tagged as *fake*) that indicates if the comment is fake or not.

Listing A.1: Yelp Analysis Example


```
{
  "@context": "http:\/\/schema.org",
  "name": "Taqueria Cazadores",
  "image": "http:\/\/s3-media3.fl.yelpcdn.com\/bphoto\/MALKEtLaRCmaphhqvvE1ow\/ls.jpg",
  "reviews": [
    {
      "@type": "Review",
      "datePublished": "12\/11\/2015",
      "ratingValue": 4,
      "reviewBody": "it was just ok, I expected more for the price though.",
      "fakeAnalysis": {
        "fake": true
      },
      "author": {
        "@type": "Person",
        "name": "Bapir A."
      }
    },
    {
      "@type": "Review",
      "datePublished": "7\/28\/2015",
      "ratingValue": 4,
      "reviewBody": "Highly recommendation of the place as have some of the good job
          here for I love coming here. last time ordered three items with my friend was
          a enjoying time here.",
      "fakeAnalysis": {
        "fake": false
      },
      "author": {
        "@type": "Person",
        "name": "Mable I."
      }
    }
  ],
  "loading": false,
  "error": null
}
```


A.2 Amazon

The crawling system recollect the same information from Amazon than Yelp. This kind of analysis is *Emotions*. The sentiment analysis is under *emotionAnalysis* key. *hasEmotionCategory* key from *ONYX* structure gives us the information of what emotion predominates in each comment.

Listing A.2: Amazon Analysis Example


```
{
  "@context": "http:\/\/schema.org",
  "name": "HP K1500 Wired Keyboard",
  "image": "http:\/\/ecx.images-amazon.com\/images\/I\/410Iim0BKaL._AC_AA60_SCLZZZZZZZ__
      .jpg",
  "reviews": [
    {
      "@type": "review",
      "datePublished": "on November 18, 2014",
      "ratingValue": 5,
      "reviewBody": "This keyboard feels solid and looks good too. Pleased.",
      "author": {
        "@type": "Person",
        "name": "DanMBA"
      },
      "emotionAnalysis": {
        "@id": "_:b0",
        "analysis": [
          {
            "version": "0.1",
            "@id": "emotext",
            "maxPolarityValue": 1,
            "name": "EmoText",
            "minPolarityValue": 0
          }
        ],
        "entries": [
          {
            "text": "This keyboard feels solid and looks good too. Pleased.",
            "@id": "Entry",
            "nif:language": "en",
            "nif:emotionSets": {
              "@id": "Emotions0",
              "emotions": [
                {
                  "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                      #valence": 7.63,
                  "onyx:hasEmotionCategory": "http:\/\/gsi.dit.upm.es\/ontologies\/
                      wnaffect\/ns#joy",
                  "@id": "Emotion0",
                  "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                      #dominance": 6.35,
                  "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                      #arousal": 5.555
                }
              ]
            }
          }
        ]
```

```
      }
    },
    {
      "@type": "review",
      "datePublished": "on March 1, 2014",
      "ratingValue": 5,
      "reviewBody": "This is a perfect replacement keyboard if you do not want to spent
          much money.  It is a steal for the price.",
      "author": {
        "@type": "Person",
        "name": "KODEN"
      },
      "emotionAnalysis": {
        "@id": "_:b0",
        "analysis": [
          {
            "version": "0.1",
            "@id": "emotext",
            "maxPolarityValue": 1,
            "name": "EmoText",
            "minPolarityValue": 0
          }
        ],
        "entries": [
          {
            "text": "This is a perfect replacement keyboard if you do not want to spent
                much money. It is a steal for the price.",
            "@id": "Entry",
            "nif:language": "en",
            "nif:emotionSets": {
              "@id": "Emotions0",
              "emotions": [
                {
                  "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                      #valence": 5.09,
                  "onyx:hasEmotionCategory": "http:\/\/gsi.dit.upm.es\/ontologies\/
                      wnaffect\/ns#anger",
                  "@id": "Emotion0",
                  "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                      #dominance": 5.144,
                  "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                      #arousal": 5.456
                }
              ]
            }
          }
        ]
      }
    }
  ], "loading": false,
  "error": null
}
```

A.3 Twitter

This result shows a sentiment analysis. The sentiment analysis is under *sentimentAnalysis* key. *hasPolarityValue* key from *MARL* structure gives us the information about the polarity of the sentiment the user shows in his comment, being the polarity value of 1 the most positive and the polarity value of 0 the most negative.

Listing A.3: Twitter Analysis Example


```
{
  "@context": "http:\/\/schema.org",
  "reviews": [
    {
      "@type": "Review",
      "datePublished": "Mon Feb 01 22:39:23 +0000 2016",
      "reviewBody": "Hab\u00eda que probarlo. goikogrill #goiko #goikogrill #burguers #
          hamburguesas #americanfood #chamberi",
      "author": {
        "@type": "Person",
        "name": "Victoria"
      },
      "sentimentAnalysis": {
        "@id": "_:b0",
        "analysis": [
          {
            "version": "0.1",
            "@id": "sentiment140_0.1",
            "maxPolarityValue": 1,
            "name": "sentiment140",
            "minPolarityValue": 0
          }
        ],
        "entries": [
          {
            "text": "Hab\u00eda que probarlo. goikogrill ",
            "@id": "Entry0",
            "nif:language": "auto",
            "opinions": [
              {
                "marl:hasPolarityValue": 0.5,
                "prov:wasGeneratedBy": "sentiment140_0.1",
                "@id": "Opinion0",
                "marl:hasPolarity": "marl:Neutral"
              }
            ]
          }
        ]
      }
```

```
    },
    {
      "@type": "Review",
      "datePublished": "Sun Jan 31 11:51:49 +0000 2016",
      "reviewBody": "RT @ErDiegoNieto: @DanieloviedoM a la bater\u00eda de #Goiko en
          @truiteatre #Mallorca #entradasagotadas. Vente paca @dominguezja",
      "author": {
        "@type": "Person",
        "name": "Lucia\u2665"
      },
      "sentimentAnalysis": {
        "@id": "_:b0",
        "analysis": [
          {
            "version": "0.1",
            "@id": "sentiment140_0.1",
            "maxPolarityValue": 1,
            "name": "sentiment140",
            "minPolarityValue": 0
          }
        ],
        "entries": [
          {
            "text": "RT @ErDiegoNieto: @DanieloviedoM a la bater\u00eda de ",
            "@id": "Entry0",
            "nif:language": "auto",
            "opinions": [
              {
                "marl:hasPolarityValue": 0.5,
                "prov:wasGeneratedBy": "sentiment140_0.1",
                "@id": "Opinion0",
                "marl:hasPolarity": "marl:Neutral"
              }
            ]
          }
        ]
      }
    }
  ],
  "hashtag": "#goiko",
  "loading": false,
  "error": null
}
```

A.4 Facebook

This result shows a multiple analysis: Fake, Sentiments and Emotions.

Listing A.4: Facebook Analysis Example


```
{
  "@context": "http:\/\/schema.org",
  "name": "Goiko Grill",
  "image": "https:\/\/scontent.xx.fbcdn.net\/hprofile-xtl1\/v\/t1.0-1\/p200x200
      \/11742653_476490885844148_4161450282861413292_n.jpg?oh=4291
      d729434f02272b03c13ce6a5fda1&oe=57420DEF",
  "reviews": [
    {
      "datePublished": "2015-12-21T19:46:22+0000",
      "author": {
        "@type": "Person",
        "name": "Marcos de la Osa"
      },
      "reviewBody": "\u00a1Y est\u00e1 muy buena!",
      "fakeAnalysis": {
        "fake": false
      },
      "sentimentAnalysis": {
        "@id": "_:b0",
        "analysis": [
          {
            "version": "0.1",
            "@id": "sentiment140_0.1",
            "maxPolarityValue": 1,
            "name": "sentiment140",
            "minPolarityValue": 0
          }
        ],
        "entries": [
          {
            "text": "\u00a1Y est\u00e1 muy buena!",
            "@id": "Entry0",
            "nif:language": "auto",
            "opinions": [
              {
                "marl:hasPolarityValue": 0.5,
                "prov:wasGeneratedBy": "sentiment140_0.1",
                "@id": "Opinion0",
                "marl:hasPolarity": "marl:Neutral"
              }
            ]
          }
        ]
      },
```

```
    "emotionAnalysis": {
      "@id": "_:b0",
      "analysis": [
        {
          "version": "0.1",
          "@id": "emotext",
          "maxPolarityValue": 1,
          "name": "EmoText",
          "minPolarityValue": 0
        }
      ],
      "entries": [
        {
          "text": "\u00a1Y est\u00e1 muy buena!",
          "@id": "Entry",
          "nif:language": "en",
          "nif:emotionSets": {
            "@id": "Emotions0",
            "emotions": [
              {
                "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                    #valence": 0,
                "onyx:hasEmotionCategory": "http:\/\/gsi.dit.upm.es\/ontologies\/
                    wnaffect\/ns#neutral-emotion",
                "@id": "Emotion0",
                "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                    #dominance": 0,
                "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                    #arousal": 0
              }
            ]
          }
        }
      ]
    }
  },
  {
    "datePublished": "2016-02-02T21:22:05+0000",
    "author": {
      "@type": "Person",
      "name": "Fredy Luis Goncalves Barbosa"
    },
    "reviewBody": "En hora buena, y que sigan los \u00e9xitos",
    "fakeAnalysis": {
      "fake": false
    },
    "sentimentAnalysis": {
      "@id": "_:b0",
      "analysis": [
        {
          "version": "0.1",
          "@id": "sentiment140_0.1",
```

```
          "maxPolarityValue": 1,
          "name": "sentiment140",
          "minPolarityValue": 0
        }
      ],
      "entries": [
        {
          "text": "En hora buena, y que sigan los \u00e9xitos",
          "@id": "Entry0",
          "nif:language": "auto",
          "opinions": [
            {
              "marl:hasPolarityValue": 0.5,
              "prov:wasGeneratedBy": "sentiment140_0.1",
              "@id": "Opinion0",
              "marl:hasPolarity": "marl:Neutral"
            }
          ]
        }
      ]
    },
    "emotionAnalysis": {
      "@id": "_:b0",
      "analysis": [
        {
          "version": "0.1",
          "@id": "emotext",
          "maxPolarityValue": 1,
          "name": "EmoText",
          "minPolarityValue": 0
        }
      ],
      "entries": [
        {
          "text": "En hora buena, y que sigan los \u00e9xitos",
          "@id": "Entry",
          "nif:language": "en",
          "nif:emotionSets": {
            "@id": "Emotions0",
            "emotions": [
              {
                "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                    #valence": 0,
                "onyx:hasEmotionCategory": "http:\/\/gsi.dit.upm.es\/ontologies\/
                    wnaffect\/ns#neutral-emotion",
                "@id": "Emotion0",
                "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                    #dominance": 0,
                "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                    #arousal": 0
              }
            ]
```

```
            }
          }
        ]
      }
    }
  ],
  "loading": false,
  "error": null
}
```


A.5 YouTube

The fields of this result are the same of Yelp or Amazon, explained before.

Listing A.5: YouTube Analysis Example


```
{
  "@context": "http:\/\/schema.org",
  "name": "eW3gMGqcZQc",
  "image": "http:\/\/img.youtube.com\/vi\/eW3gMGqcZQc\/maxresdefault.jpg",
  "reviews": [
    {
      "@type": "Review",
      "date": "2016-01-27T16:08:36.484Z",
      "reviewBody": "I hate trying to learn stuff online...I am very hands on and visual
          , in \naddition when I ask a question I want an answer I don't want to wait 5
          \nminutes or 5 hours or 5 days for an answer.... by the time I wait for \
          nsomeone to type an answer ill usually have moved on or away from what I was \
          ntrying to figure out....\ufeff",
      "author": {
        "@type": "Person",
        "name": "P4L9BLACK"
      },
      "emotionAnalysis": {
        "@id": "_:b0",
        "analysis": [
          {
            "version": "0.1",
            "@id": "emotext",
            "maxPolarityValue": 1,
            "name": "EmoText",
            "minPolarityValue": 0
          }
        ],
        "entries": [
          {
            "text": "I hate trying to learn stuff online...I am very hands on and visual
                , in \\naddition when I ask a question I want an answer I don't want to
```

```
            wait 5 \\nminutes or 5 hours or 5 days for an answer.... by the time I
            wait for \\nsomeone to type an answer ill usually have moved on or away
            from what I was \\ntrying to figure out....\\ufeff",
          "@id": "Entry",
          "nif:language": "en",
          "nif:emotionSets": {
            "@id": "Emotions0",
            "emotions": [
              {
                "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                    #valence": 5.6981818181818,
                "onyx:hasEmotionCategory": "http:\/\/gsi.dit.upm.es\/ontologies\/
                    wnaffect\/ns#anger",
                "@id": "Emotion0",
                "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                    #dominance": 5.2845454545455,
                "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                    #arousal": 5.3472727272727
              }
            ]
          }
        }
      ]
    }
  },
  {
    "@type": "Review",
    "date": "2015-09-24T04:33:36.528Z",
    "reviewBody": "nice video\ufeff",
    "author": {
      "@type": "Person",
      "name": "Rohini Komarappagari"
    },
    "emotionAnalysis": {
      "@id": "_:b0",
      "analysis": [
        {
          "version": "0.1",
          "@id": "emotext",
          "maxPolarityValue": 1,
          "name": "EmoText",
          "minPolarityValue": 0
        }
      ],
      "entries": [
        {
          "text": "nice video\\ufeff",
          "@id": "Entry",
          "nif:language": "en",
          "nif:emotionSets": {
            "@id": "Emotions0",
            "emotions": [
```

```
                {
                  "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                      #valence": 6.55,
                  "onyx:hasEmotionCategory": "http:\/\/gsi.dit.upm.es\/ontologies\/
                      wnaffect\/ns#joy",
                  "@id": "Emotion0",
                  "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                      #dominance": 5.58,
                  "http:\/\/www.gsi.dit.upm.es\/ontologies\/onyx\/vocabularies\/anew\/ns
                      #arousal": 4.38
                }
              ]
            }
          }
        ]
      }
    }
  ],
  "loading": false,
  "error": null
}
```


A.6 TripAdvisor

The fields of this result are the same of Yelp or Amazon, explained before.

Listing A.6: TripAdvisor Analysis Example


```
{
  "name": "Casa Flores",
  "image": "http:\/\/media-cdn.tripadvisor.com\/media\/photo-s\/09\/47\/bf\/6f\/getlstd-
      property-photo.jpg",
  "reviews": [
    {
      "@type": "Person",
      "datePublished": "Reviewed November 20, 2015",
      "ratingValue": 5,
      "reviewBody": "The has been of the best dinner that I have ever had! Everything
          was perfect and the food was AMAZING!!!",
      "author": {
        "@type": "Person",
        "name": "Darya F"
      },
      "sentimentAnalysis": {
        "@id": "_:b0",
        "analysis": [
          {
            "version": "0.1",
```

```
          "@id": "sentiment140_0.1",
          "maxPolarityValue": 1,
          "name": "sentiment140",
          "minPolarityValue": 0
        }
      ],
      "entries": [
        {
          "text": "The has been of the best dinner that I have ever had! Everything
              was perfect and the food was AMAZING!!!",
          "@id": "Entry0",
          "nif:language": "auto",
          "opinions": [
            {
              "marl:hasPolarityValue": 1,
              "prov:wasGeneratedBy": "sentiment140_0.1",
              "@id": "Opinion0",
              "marl:hasPolarity": "marl:Positive"
            }
          ]
        }
      ]
    }
  },
  {
    "@type": "Person",
    "datePublished": "Reviewed November 18, 2015",
    "ratingValue": 5,
    "reviewBody": "Vanessa and her husband are the warmest hosts. They make you feel
        at home. The food is delicious! All of it, but Vanessa's specialty are her
        homemade pastas. They steal the show!",
    "author": {
      "@type": "Person",
      "name": "Liz S"
    },
    "sentimentAnalysis": {
      "@id": "_:b0",
      "analysis": [
        {
          "version": "0.1",
          "@id": "sentiment140_0.1",
          "maxPolarityValue": 1,
          "name": "sentiment140",
          "minPolarityValue": 0
        }
      ],
      "entries": [
        {
          "text": "Vanessa and her husband are the warmest hosts. They make you feel
              at home. The food is delicious! All of it, but Vanessa's specialty are
              her homemade pastas. They steal the show!",
          "@id": "Entry0",
```

```
            "nif:language": "auto",
            "opinions": [
              {
                "marl:hasPolarityValue": 0.5,
                "prov:wasGeneratedBy": "sentiment140_0.1",
                "@id": "Opinion0",
                "marl:hasPolarity": "marl:Neutral"
              }
            ]
          }
        ]
      }
    }
  ],
  "loading": false,
  "error": null
}
```

APPENDIX B

Instructions for deploying GSI Crawler

This chapter aims to provide a general walk-trough on how to deploy our GSI Crawler system. It assumes basic knowledge on both the system and the tools used, and the required tools.

B.1 Preparing the environment

GSI Crawler consists of several modules. They have been packed together, except *Senpy*[1] which is an external and independent module. The system needs specific dependencies to work. In this section the installation of those dependencies will be covered. These instructions are suposed you are running a *Debian based* Linux distribution, specifically *Ubuntu 14.04 x64* but it might work with other distributions.

[1] https://github.com/gsi-upm/senpy

B.1.1 Installing dependencies

Listing B.1: Installing dependencies

```
# We need Python to exec our code
sudo apt-get install python

# Python package manager to install Python libraries
sudo apt-get install python-pip

# Browser controller from Python script
sudo pip install selenium
# Headless browser needed to build TripAdvisor
sudo apt-get install phantomjs

# Library to deploy a WSGI for Python
sudo pip install bottle

# Crawling Scrapy Framework
sudo pip install scrapy

# Twitter API Client
pip install twython

# Youtube API Client
sudo pip install --upgrade google-api-python-client

# Other Python libraries needed
sudo pip install parse
sudo pip install pytz
```

B.2 Deploying Senpy

The most simple way to deploy *Senpy* module is using a Docker Image. Docker is needed to be installed before this step. You can follow official Docker installing instructions at Docker Official Website[2]. Once Docker is installed in your system, *Senpy* module can be downloaded and executed running the following command (Listing C.1).

Listing B.2: Deploying Senpy

```
docker run -ti -p 5000:5000 balkian/senpy --host 0.0.0.0 --default-plugins
```

[2]https://docs.docker.com/engine/installation/ubuntulinux/

B.3 Running GSI Crawler

Once dependencies are installed and Senpy is deployed, GSI Crawler server can be started using the following command (Listing B.3) at GSI Crawler root folder.

Listing B.3: Running GSI Crawler

```
python web.py
```

Now GSI Crawler can be accessed using the following url: `http://localhost:8888/`.

APPENDIX C

Deploying GSI Crawler using Docker

This chapter aims to provide a general walk-trough on how to deploy our GSI Crawler system. It assumes basic knowledge on both the system and the tools used, and the required tools.

The most simple way to deploy *GSI Crawler* platform is using a Docker Image. Docker is needed to be installed before this step. You can follow official Docker installing instructions at Docker Official Website[1]. Once Docker is installed in your system, *GSI Crawler* platform can be downloaded and deployed running the following command (Listing C.1).

Listing C.1: Deploying Senpy

```
docker run -ti -p 8888:8888 pepos/gsicrawler --host 0.0.0.0
```

Now GSI Crawler can be accessed using the following url: `http://localhost:8888/`.

[1] https://docs.docker.com/engine/installation/ubuntulinux/