UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DEVELOPMENT OF A YOUTUBE VIDEOS FEELINGS ANALISER

José Miguel del Valle Salas

2017

TRABAJO FIN DE GRADO

Título:	Desarrollo de un analizador de sentimientos de videos de Youtube
Título (inglés):	Development of a YouTube videos feelings analiser.
Autor:	José Miguel del Valle Salas
Tutor:	Juan Fernando Sánchez Rada
Ponente:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:

Vocal:

Secretario:

Suplente:

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO FIN DE GRADO

DEVELOPMENT OF A YOUTUBE VIDEOS FEELINGS ANALISER

José Miguel del Valle Salas

Julio de 2017

Resumen

YouTube es una de las redes sociales más exitosas de la actualidad, por lo que tiene cada vez más peso en nuestra sociedad. Debido a esto resulta de gran utilidad conocer los sentimientos que provocan en el público los diferentes videos que esta plataforma ofrece.

Este proyecto se ha basado en el desarrollo de una herramienta capaz de analizar estos sentimientos, la cual podrá ser utilizada con diversos fines como estudios de mercado o el aprendizaje emocional en personas con cierta diversidad funcional.

Para el desarrollo del proyecto se han utilizado las siguientes tecnologías:

- Luigi (Extracción, formateo, análisis con senpy y envío a Elasticsearch de los datos.)
- D3.js (Visualización de los datos analizados.)
- Polymer Web Components (Creación de widgets reutilizables para visualizar el análisis.)

El resultado final ha sido la creación de un dashboard integrable con sefarad, que contiene 3 widgets:

1) yt-video-widget: Muestra el video que se va a analizar.
2) yt-sentiment-widget: Muestra una gráfica con los resultados del análisis de los subtítulos del video.
3) yt-comments-widget: Muestra los comentarios del vídeo clasificados según el sentimiento que provoca.

Palabras clave: YouTube, Videos, Sentimientos, D3.js, Luigi, Senpy, Polymer, Widgets.

Abstract

Nowadays, Youtube is one of the most successful social networks, therefore it has more and more impact in our society.

Due to this it's quite useful to know the sentiments that this platform videos produces.

This project has been focused in the development of a tool able to analise this sentiments, which could be used for different purposes like Market studies or emotional learning for people who has some functional diversity.

The technologies used during the project development has been:

- Luigi (Data extraction, formatting, analysis with senpy and sending to Elasticsearch.)
- D3.js (Analysed data displaying.)
- Polymer Web Components (Creation of reusable widgets for analysis displaying.)

The final result has been the creation of a sefarad-integrable dashboard which contains 3 widgets:

1) yt-video-widget: Displays the video which is going to be analised.
2) yt-sentiment-widget: Displays a graph with the results of the video subtitles analysis.
3) yt-comments-widget: Displays the video comments classified by the sentiment produced.

Keywords: YouTube, Videos, Sentiments, D3.js, Luigi, Senpy, Polymer, Widgets.

Agradecimientos

A lo largo de mi carrera me he encontrado con múltiples baches. Si hubiese estado sólo, no podría haberlos superado. Por ello quiero agradecer a todos aquellos que han conseguido que siga adelante.

Le quiero dar las gracias a mis padres, por haberme apoyado y confiar en mí siempre.

A mi hermana, por todas las veces que estando agobiado se ha quedado conmigo para que pueda despejarme la cabeza, y sobretodo, por las risas que nos echamos siempre.

A todos mis amigos, por el apoyo incondicional que me habéis dado todos estos años, por haberme ayudado cuando lo he necesitado, por haberme consolado cuando estaba mal (incluso los que están a cientos de kilómetros de distancia), en fin, por ser los mejores amigos del mundo, muchísimas gracias a todos.

A mi prima Chari, por haber sido mi ejemplo a seguir y por haberme encaminado a teleco.

A todos los miembros del club de teatro NoEsCulpaNuestra (mención especial a Jack, Desbarbado, por permitirme incluir imágenes de uno de sus videos en el presente trabajo), por haberme abierto el mundo en general.

A mis tutores J e Iglesias, por haberme aguantado tanto tiempo y haberme encaminado para mejorar cada vez más, no sólo en mi TFG, sino en mi vida, muchas gracias por todo.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Context

YouTube has become one of the most popular social networks since its creation in 2005. We can find millions of videos and comments created by the community, which is increasing every year. In 2017, it has more than 1000 million users around the world (almost 1/3 of the total internet users.)[10]

With this numbers, there's no surprise in the fact that many enterprises has put a lot of interest in this platform and its creators for marketing purposes. This has been a hot topic in the past months due to a scandal produced because of a series of YouTube videos, which content was not adequate, having some ads of many brands. This has produced a radical change in the way YouTube selects the videos which can contain ads[1].

In this project, we are going to develop a useful tool that could help with the above problem, which can display the sentiments that videos produce in the public, by analysing their subtitles and comments. This will be a quite useful data source for enterprises in order to select YouTube creators for their ads.

To do so, we need three components that will compose the tool:

- A YouTube video displayer.
- A graph generator to display the video subtitles sentiments.
- A comments classifier depending the sentiments reflected by them.

We will use Polymer in order to make the needed components. We will gather them creating a Polymer dashboard that will be the mentioned tool.

In order to get the necessary data for the analysis, we'll use Luigi pipelines that will get the subtitles, comments and the ID from a video.

1.2 Project goals

In the long term, this project aim at displaying sentiment data about YouTube videos. With this aim, the project will collect subtitles and comments from YouTube videos and will analyse them using senpy and will display the collected data with a developed dashboard containing three widgets.

Among the main goals inside this project, we can find:

- Collect subtitles and comments from a youtube video.
- Build a pipeline in order to analyse the data.
- Develop a dashboard containing three widgets that displays all the analysed data.

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 explains the context in which this project is developed. Moreover, it describes the main goals to achieve in this project.
Chapter 2 provides a description of the main technologies on which this project relies.
Chapter 3 describes the architecture of this project.
Chapter 4 exposes the developing process of the system components.
Chapter 5 presents the project main use case.
Chapter 6 discusses the conclusions drawn from this project, problems faced and suggestions for a future work.

CHAPTER 2

Enabling Technologies

2.1 Introduction

In this chapter we are going to give an insight into the technologies used in this project. First of all we are going to explain Polymer, the technology used to build the different components (widgets) developed in this project. Then we are going to give an approach to D3.js, a library used to display the graph that will include the subtitles analysed data, elasticsearch, used to add and search data, Senpy, a text-based sentiment analyser used to analyse our data, Docker, used to contain and run our system, bower, used to generate accessible dependency packages composed by our widgets, and Luigi, used to get, analyse (Using Senpy) and store data into elasticsearch.

2.2 Polymer web components

Polymer is a library developed by Google which function is to create web components. This components are easily maintainable, reusable, custom elements. It's compatible with multiple browsers and libraries, which allows us to create complex web apps without compatibility problems.

This components are quite useful in order to build different applications with similar elements. You only have to define a component once and then use it in all the projects you want setting the parameters defined.

Polymer uses HTML, JavaScript and CSS to define web components:

- HTML is used to define the component structure creating a template for it.
- CSS is used to give style to the element.
- JavaScript is used to make the elements interactive.

Here we have an example of a Polymer Web Component:[5]

```
<dom-module id="element-name">

  <template>
    <style>
      /* CSS rules for your element */
    </style>

    <!-- local DOM for your element -->

    <div>{{greeting}}</div> <!-- data bindings in local DOM -->
  </template>

  <script>
    // element registration
    Polymer({
      is: "element-name",

      // add properties and methods on the element's prototype

      properties: {
        // declare properties for the element's public API
        greeting: {
          type: String,
          value: "Hello!"
        }
      }
    });
  </script>

</dom-module>
```

2.2.1 Polymer catalog

Polymer gives a collection of pre-built elements that can be put in out page and be usable inmediately. This elements are grouped by categories. This categories provide us a series of components that makes easier to develop our web applications. Between them we can found:

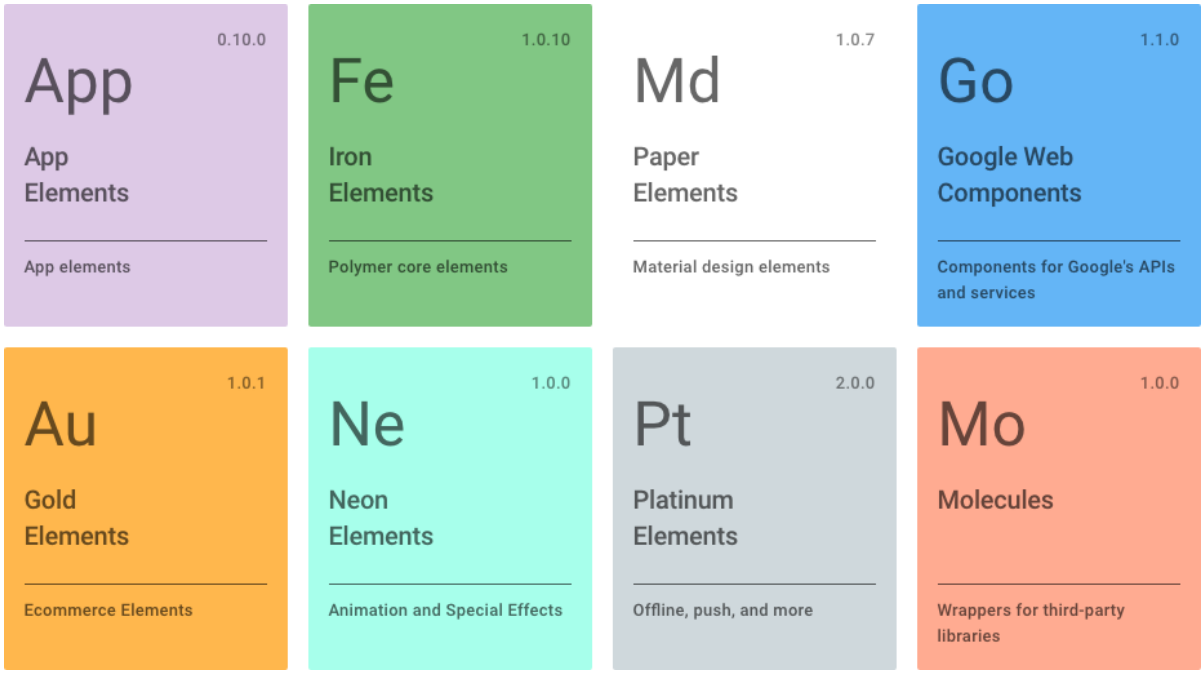

Figure 2.1: Polymer element categories

- App Elements: Elements that helps to build entire applications providing features like routing, data storage, internationalization, layout elements...
- Iron Elements: Provides building blocks like iron ajax, to make ajax requests, iron collapse, to make collapsible elements, iron icon, to display icons...
- Paper Elements: A set of UI components used to implement Google's material design and its behaviors, ripple effects and styles.
- Google Web Components: Here we can find a lot of Google services integrated with polymer like maps, analytics, calendar, chart...
- Gold Elements: Used for e-commerce use-cases like the usage of credit cards or mobile phone validations.
- Neon Elements: Used to implement animated transitions for Polymer Elements with web animations.

- Platinum Elements: Provides web apps features like push notifications, redirections, bluetooth, offline usage...

- Molecules: Wraps other libraries to make easier to use them with polymer. By the moment there are only 2 libraries: Marked and Prism.

2.3 D3.js

D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using standard web technologies HTML, SVG, and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation[11]

With D3 you can create all kind of document elements like graphs, bar charts, tables... using data as an input like an array of elements. This elements can manipulate a huge amount of data and have a fast response to data changes.

It's important to note that this elements are reusable so you can define a graph for a web app and use that code to display countless kinds of data with it.

We can see an example of what D3 can do in Fig. 2.2.

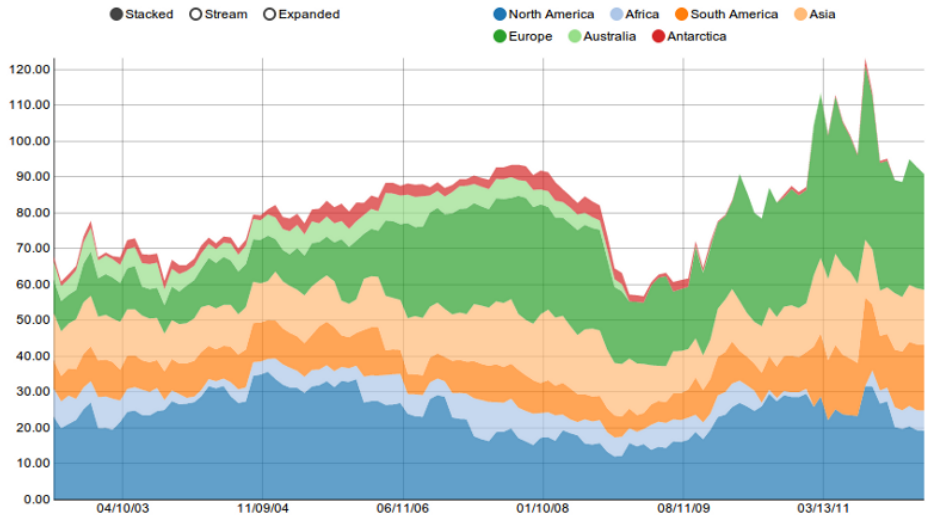

Figure 2.2: D3.js example

2.3.1 D3.js usage

D3.js works with arbitrary sets of nodes called selections. Selectors, defined by the W3C Selectors API, allow us to select elements using attribute values, classes, ID's... With this selectors we can modify the elements attributes, HTML, styles, add Event Listeners etc. In order to manipulate data, D3 implements two main selectors: enter and exit. With enter, you can create new nodes to manage the incoming data used for an element. With exit you can remove data from the data array.

Here is an example of a simple D3.js element:[9]

Listing 2.1: D3.js example

```
<!DOCTYPE html>
<html>
<head>
    <script type="text/javascript" src="http://mbostock.github.com/d3/d3.js
        "></script>
</head>
<body>
    <div id="viz"></div>
    <script type="text/javascript">
    var sampleSVG = d3.select("#viz")
        .append("svg")
        .attr("width", 100)
        .attr("height", 100);
    sampleSVG.append("circle")
        .style("stroke", "gray")
        .style("fill", "white")
        .attr("r", 40)
        .attr("cx", 50)
        .attr("cy", 50)
        .on("mouseover", function(){d3.select(this).style("fill", "
            aliceblue");})
        .on("mouseout", function(){d3.select(this).style("fill", "white")
            ;});
    </script>
</body>
</html>
```

This example displays a circle that gets highlited when we put the mouse on it. As we can see, the first step is selecting the DOM element with the "viz" id. Then it appends an svg element to it with its width and height attributes.

Then it appends the circle, defining its style, radius (r), "x" position (cx) and "y" position (cy). In the end it set the mouseover and mouseout functions, which defines the circle behavior.

2.4 Elasticsearch

Elasticsearch is an open source search engine built on top of Apache Lucene, a full text search-engine library. Lucene is arguably the most advanced, high-performance, and fully featured search engine library in existence today. Elasticsearch is written in Java and uses Lucene internally for all of its indexing and searching, but it aims to make full-text search easy by hiding the complexities of Lucene behind a simple, coherent, RESTful API.[2]

Elasticsearch is used in our project as a NoSQL database in order to store sentiment analysed data and use this data to fill the system components with it (due the fact that Polymer is compatible with Elasticsearch).

2.4.1 Elasticsearch usage

We are going to give a simple example of the Elasticsearch usage.

First of all, we are going to add some data into an Elasticsearch index. For example we can use the following command using the Elasticsearch's bulk API:

```
$ curl -s -XPOST localhost:9200/yourdata/data/_bulk --data-binary
```

We can search data using a query by two main methods depending on how we use the query:

- As a request body:

```
http://localhost:9200/yourdata/data/_search' -d '{
    "query" : {
        "term" : { "text" : "Element" }
    }
}
```

- As an URL parameter:

```
http://localhost:9200/yourdata/data/_search?q=text:Element
```

2.5 Senpy

Senpy is a Python server for sentiment analysis developed by the Intelligent Systems Group. (UPM, Madrid.) It implements an API that makes easy to use senpy with any system. Senpy can be called using different parameters, we are going to see some of the most important:

- **algorithm (algo):** Defines the algorithm used for the analysis.
- **input (i):** The text that is going to be analysed.
- **informat (f) and outformat (o):** Defines the format for the input data.
- **outformat (o):** Defines the format for the output data.

A functional demo of Senpy and sentiText can be found at [8]

2.6 Docker

Docker is an open source project that provides automatized applications deployment inside of software containers. This containers can be run using a simple Linux instance, avoiding the setup and maintenance of virtual machines. Each docker container is independent to make easier to build different apps without dependencies incompatibilities. However, Docker containers can interact with each other using Docker-compose, a tool for defining and running multi-container Docker applications.

2.6.1 Docker Compose usage

First of all, we need to define the different Dockerfiles for each desired container. In this Dockerfiles we have to set the building instructions for the container.

Then we have to make a doker-compose.yml file to establish the containers and relations set for the desired application as seen in the following example:

Listing 2.2: docker-compose.yml example

```
version: "3.3"
services:
```

```
  wordpress:
    image: wordpress
    ports:
      - 8080:80
    networks:
      - overlay
    deploy:
      mode: replicated
      replicas: 2
      endpoint_mode: vip
  mysql:
    image: mysql
    volumes:
       - db-data:/var/lib/mysql/data
    networks:
       - overlay
    deploy:
      mode: replicated
      replicas: 2
      endpoint_mode: dnsrr
volumes:
  db-data:
networks:
  overlay:
```

Finally, we only have to build and run the containers by using:

```
sudo docker-compose up
```

2.7 Bower

Bower is an open source dependency packages manager that uses connections to repositories to generate easily accessible dependency packages that can be included in any application by including desired package name and its version in a file called bower.json and executing the following command.

```
sudo bower install --allow-root
```

In our system we used bower in order to make our widgets and dashboards independent and integrable components in order to make them reusable for any polymer component/application.

2.8 Luigi

Luigi is a Python module that can build complex pipelines of batch jobs. It handles dependency resolution, workflow management, visualization etc. Luigi uses a script that contains the pipeline to follow. The pipeline is based on tasks with an execution sequence. We only have to call the last task with Luigi because this one will have the data obtained by the previous tasks.

Luigi provides a scheduler that shows useful data about the pipeline executions as seen in Fig. 2.3.

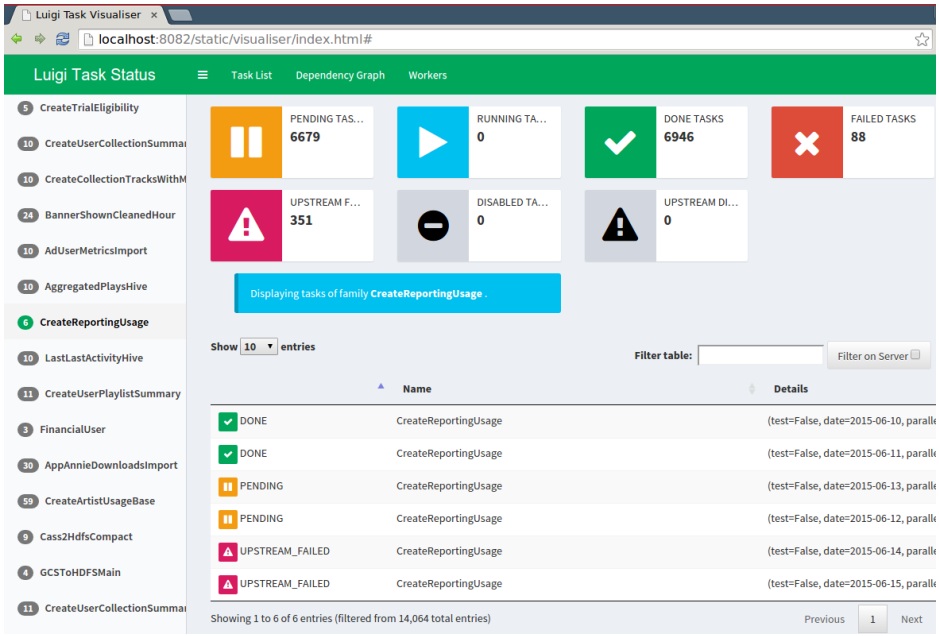

Figure 2.3: Luigi scheduler example

2.8.1 Luigi usage

There are two main concepts in Luigi:

- Targets, classes that corresponds to a file on a disk. It has an "exist" method which returns true if the target exists. We should use LocalTarget to directly map to a file on the local drive.
- Tasks, where Luigi work gets done. It has three main methods to implement: requires, output and run.

  Example:[3]

```
import luigi
class MyTarget(luigi.Target):

    def exists(self):
        """
        Does this Target exist?
        """
        return True
class MyExampleTask(luigi.Task):
    # Example parameter for our task: a
    # date for which a report should be run
    report_date = luigi.DateParameter()
    def requires(self):
        """
        Which other Tasks need to be complete before
        this Task can start? Luigi will use this to
        compute the task dependency graph.
        """
        return [MyUpstreamTask(self.report_date)]
    def output(self):
        """
        When this Task is complete, where will it produce output?
        Luigi will check whether this output (specified as a Target)
        exists to determine whether the Task needs to run at all.
        """
        return S3Target('s3://my-output-bucket/my-example-tasks-output
            ')
    def run(self):
        """
        How do I run this Task?
        Luigi will call this method if the Task needs to be run.
        """
```

These three methods we have to implement defines the execution of the task:

- The method "requires" defines the pipeline tasks that have to be done before running this task.
- The "output" method is where we have to set the output of the task it could be a JSON file, a text file...
- The run method defines the logic of the task, so we can decide what to do with the input data and its treatment.

CHAPTER 3

Architecture

3.1 Introduction

In this chapter, we will explain the architecture of this project, defining the design phase and implementation details. First of all, in the overview we will present a global vision about the project architecture, identifying the visualization server and other necessary modules. Secondly, we will focus on each module explaining its purpose in this project.

3.2 Overview

In this section we are going to explain the architecture of this project, defining the different blocks that compose the whole system.

- **Orchestrator:** Luigi is used as orchestrator in order to create pipelines able to collect data and dispatch it between the sentiment analysis service and the indexing and searching system.
- **Sentiment analysis service:** We used Senpy[7] to analyse the sentiments produced by YouTube subtitles and comments data classifying it between positive and negative

sentiments.

- **Indexing and search system:** This system is composed by Elasticseach, a search server able to index and search data in a quite efficient way.

- **Visualization:** This module is composed by three widgets that display the analysed data. This widgets are embedded in a dashboard based on Tourpedia-Dashboard from Sefarad.[6]

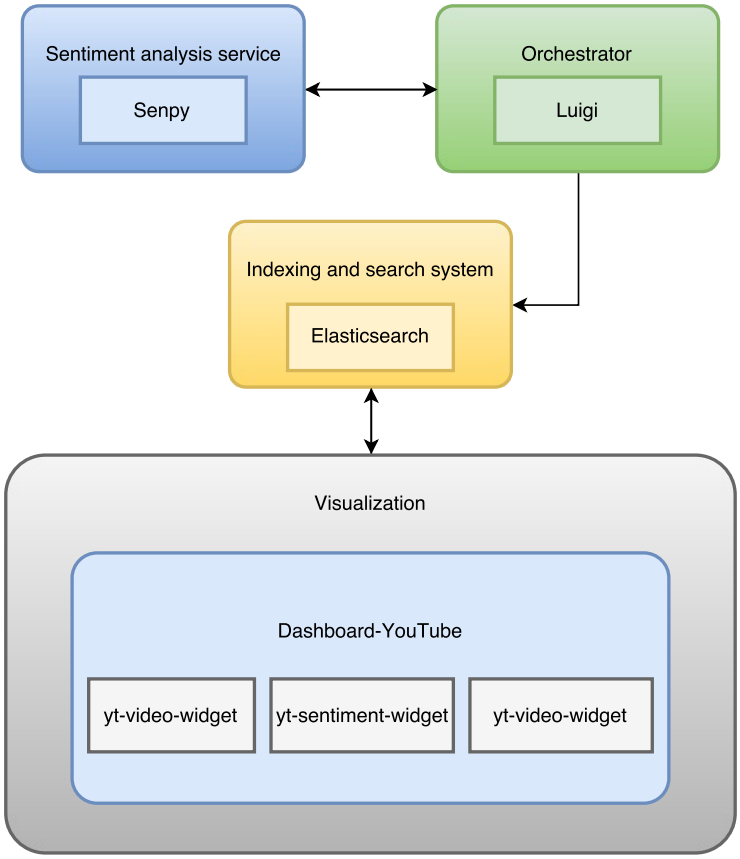

Figure 3.1: Architecture

Each element made for this project has been located in a independent Docker container.

We have used Docker Compose in order to deploy our system, using one container per widget, one container for the dashboard and two additional containers for the Elasticsearch and Luigi servers.

In the following sections we are going to deeply describe the subsystems involved in the project.

3.3 Orchestrator

We have chosen Luigi to make the Orchestator due to the efficient modulation of the different task used to manage the data, the possibility of interacting with all kind of elements like elasticsearch or Senpy, and its ease of use, allowing us to extract, manage and load data using one simple command.

We created a Luigi pipeline in order to extract, manage and load data, as we will see in the next chapter.

3.4 Sentiment analysis service

We choosed Senpy in order to analyse the data. We implemented different calls to the Senpy API in our system using sentiText, a plug-in that classifies the sentiments between positive, negative and .[4]

3.5 Indexing and search system

We are going to use Elasticsearch to index and search sentiment data due to its efficient data storage system, its persistence and its schema-free structure using JSON format.

3.6 Visualization

We have chosen Polymer to make a dashboard in order to visualize the analysed data. Polymer allows us to create different dashboards combining reusable widgets, so we can analyse several videos with the same developed widgets and integrate them in other existing dashboards.This dashboard can be integrated in Sefarad.

Sefarad is an environment developed to explore, analyse and visualize data. It is divided in three docker containers: One for Elasticsearch, another for Luigi and the last one used for visualization.

The visualization module is used to display analysed data. This module is structured in different dashboards such as Tourpedia. These dashboards contain different widgets built using Polymer Web Components.

Sefarad architecture is shown in Fig. 3.2. This architecture is based on a dashboard

that uses data taken from Twitter.

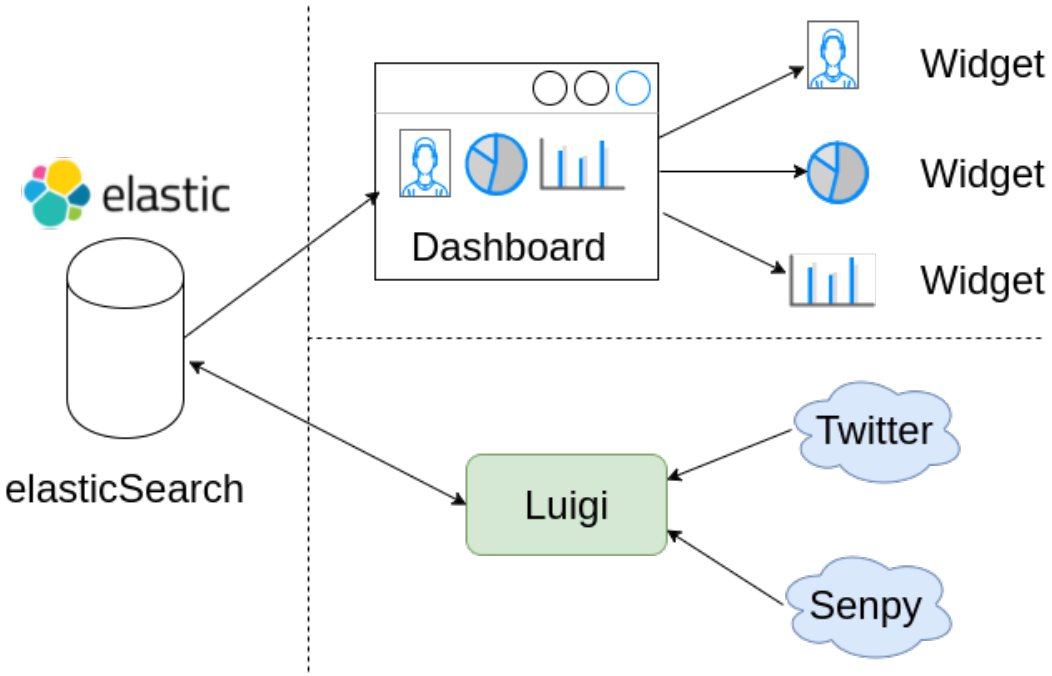

Figure 3.2: Sefarad architecture

We have develop our own dashboard named dashboard-youtube for our project. This dashboard displays data analysed from YouTube subtitles and comments using three independent and integrable widgets (This means that each widget can be used in any dashboard by including it as a bower dependency):

- **yt-video-widget:** Shows the analysed video. **yt-sentiment-widget:** Displays a graph with the subtitles sentiments data. **yt-comments-widget:** Shows two tables classifying the video comments by positive or negative sentiments.

We are going to analyse this three widgets with more detail in the next chapter.

CHAPTER 4

Dashboard development

4.1 Introduction

We are going to see in detail the dashboard-youtube development process. This dashboard will display different sentiment data from youtube videos. It is based on Polymer Web Components, a library based on Google's material design that allows the creation of reusable widgets in web applications. We have used D3.js to create a graph that displays sentiments data.[5]

4.2 Widgets development

In this section we are going to analyse the three widgets developed for the dashboard-youtube: yt-comments-widget, yt-sentiment-widget and yt-video-widget.

4.2.1 yt-comments-widget

This widget displays a table containing the video comments classifying them by positive and negative. The displayed comments can be changed using three different buttons that

shows positive comments, negative comments and all of them mixed.

First of all, we made the structure of the widget, displaying the empty table where comments will be shown. Then we used the elasticsearch index in order to take the comments that we previously loaded into elasticsearch using the luigi pipeline, and manage that data in the widget script to generate the different rows. Finally we classified and distributed the comments rows using a function that adds each one depending the visualization chosen type.

As a result we have a first widget view that displays the three buttons and a message (Fig. 4.1.).

Figure 4.1: yt-comments-widget demo 1

Then, if we click a button the two tables are shown as wee can see in Fig. 4.2.

Figure 4.2: yt-comments-widget demo 2

The size of this widget is maximized for this demo.

4.2.2 yt-sentiment-widget

This is the most complex widget we made. We created two inputs, corresponding to the initial and final time of the video we want to analyse, and a Show button that displays the data as seen in Fig. 4.3.

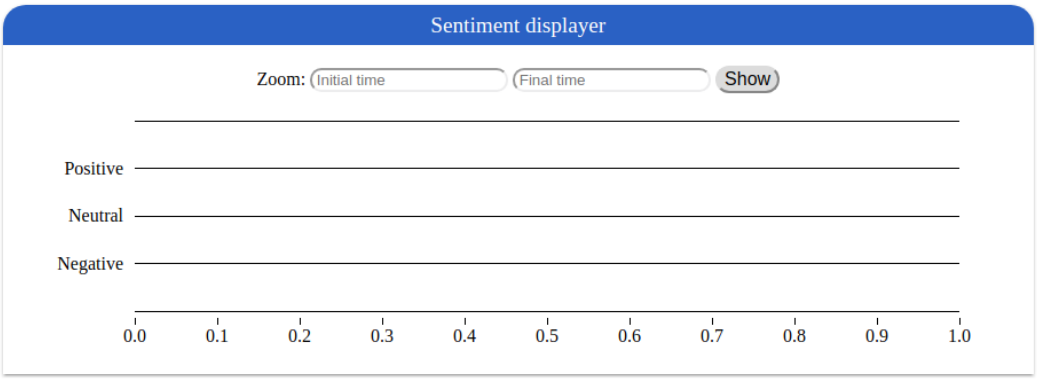

Figure 4.3: yt-sentiment-widget demo 1

This input will be used for zoom purposes. We set an interval that gets activated when the widget video is clicked, using Polymer two-way data binding, which allows us to set parameters that changes the data in all widgets when its value changes. This interval triggers a function each second that prints the graph points while the video is playing.

In order to show the data, it takes the video captions we loaded into elasticsearch, and then uses D3 to make a graph that displays different points representing the different captions sentiments related to the corresponding second for each caption.

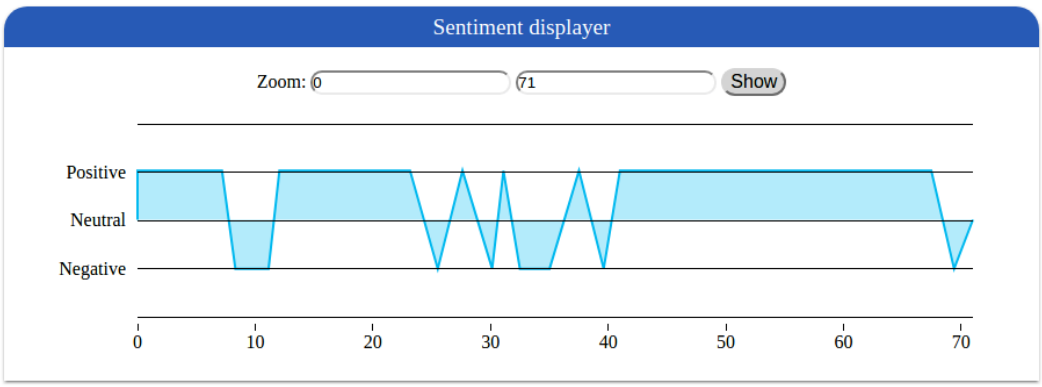

Figure 4.4: yt-sentiment-widget demo 2

When we select a point, it will be highlighted and if we click on it then the corresponding

time will be set in the video, changing the parameter used in yt-youtube-displayer to load the video.

4.2.3 yt-video-widget

This widget has been designed to display the video analysed, as we can see in Fig. 4.5.

Figure 4.5: yt-video-widget demo

This widget uses several parameters that will be given from elasticsearch loaded data. videoId is the id from the video we are going to analyse, it is embedded in a iframe that will display the video., initialtime is used to set the initial second of the video and playing to advise yt-sentiment-displayer when the video is playing.

We have chosen a minimalist design for this widget due to the needed size of the video to be properly displayed and the non-static content of it.

4.3 Dashboard development

Once the three widgets were created, we uploaded each widget into three different github repositories in order to add them to three different bower packages. By this point, we have three fully independent and integrable widgets. Each widget offers one different sentiment

data visualization and we could make a new component, including the three widgets in its bower dependencies, to obtain the three sentiment data visualization parts in one component. Due to this new component is composed by an aggregation of other components, we called this new component a dashboard.

In this section we are going to expose the steps followed in order to create this dashboard that we called youtube-dashboard.

First of all, we created a polymer template containing the three widgets we made (taking them from the bower_components created when the bower dependencies were installed), using the video widget on the top and the sentiments and comments widgets on the bottom.

We took all the elasticsearch loaded data to fill this widgets with it, using the different IDs used for each of them. The index we are going to use is "youtube". It will store sentiment data containing:

- **videoId:** Id of the video we want to analyse.
- **caption:** Part of the video subtitles text.
- **comment:** Text from the comments of the video.
- **sentiment:** Can be positive, negative or neutral.
- **polarity:** Numeric value of the sentiment. It can be -1, 0 or 1.
- **start:** Second when the text analysed starts.

Then, we created the luigi pipeline in order to fill the dashboard with data.

The main tasks in our Luigi pipeline are:

- (ExtractorTask:) This task takes the video ID from the parameters and uses youtube-dl library and the Youtube API to extract the video captions in .VTT format and a .JSON containing the comments associated to that video, and then writes this data into the output file.
- (FormatTask:) This task takes as an input the comments and captions obtained in ExtractorTask and puts them together in order to create an unique JSON containing all the data that is going to be analysed and including an id for each entry.
- (FetchDataTask:) This task reads the output data from FormatTask and modifies it in order to obtain the necessary fields for its future treatment by senpy and Elasticsearch.

It's apart from the previous task in order to make Format Task reusable with other kind of systems.

- (SenpyTask:) This task loads data fetched with previous task and send it to Senpy in order to obtain a file with the sentiments generated by the previous data. This task makes two different calls to senpy on the run(self) method depending on the data to be analysed. It analyses the "caption" field when the data comes from subtitles. In the case of comments it analyses the "comment" field.

- (Elasticsearch:) This final task loads JSON data generated by SenpyTask into an Elasticsearch index in order to make this data accessible for the visualization module. The index name is passed as a parameter when we call the Luigi pipeline.

In the end, we obtained the dashboard shown in Fig. 4.6.

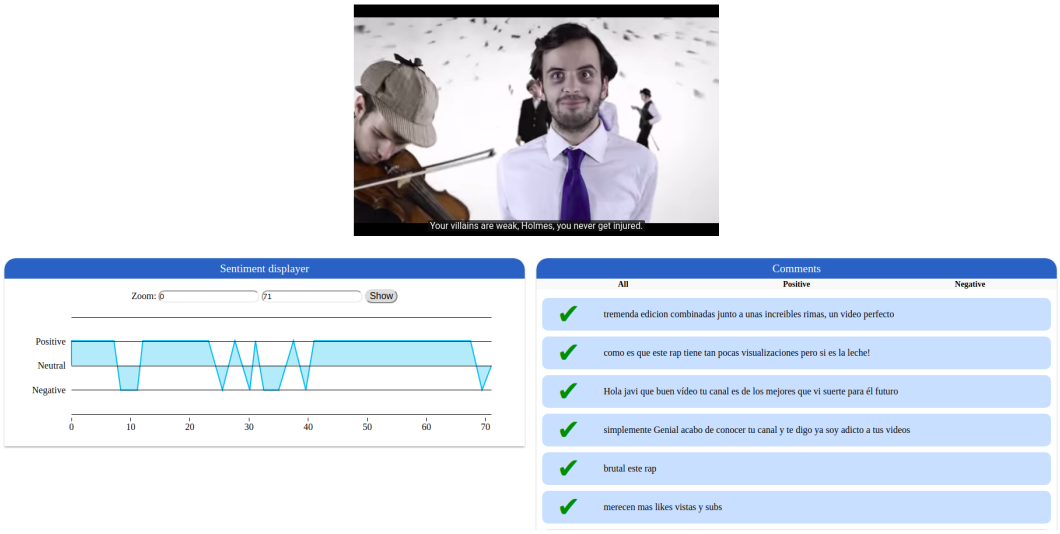

Figure 4.6: dashboard-youtube

Finally we uploaded our dashboard to another repository and then added it to a bower package, so now we could even implement our dashboard inside of another dashboard.

CHAPTER 5

Use case

5.1 Introduction

The main use case for our project could be as follows: "We are YouTube content creators and we want to know which sentiment our videos transmit, and the sentiments produced for our viewers" This chapter describes the process needed in order to run the system and visualize the analyzed data. This process will be presented as a walk-through, so anyone could reproduce it and analyse the sentiments from any desired YouTube video (if comments are available on it), so the creators and enterprises can use it to analyse YouTube sentiment data.

5.2 Deploying process

First of all, we have to obtain the dashboard. to do so, we have to clone the following repository:

`lab.cluster.gsi.dit.upm.es/tfg/TFG-JoseMiguelDelValle`

Then, we have to open a terminal and move into the project folder. In order to build

and run the dashboard, we have to use the following command:

```
sudo docker-compose up
```

Then all the docker containers included in the docker-compose.yml (dashboard, widgets, luigi and elasticsearch servers) will be built and run. Once we have all the containers running, we can access to http://localhost:8080/ to see the dashboard as seen in Fig. 5.1.

Figure 5.1: dashboard-youtube vacío

5.3 Collecting Data

We can appreciate that there is no data loaded in the dashboard. To obtain data, we have to open another terminal (the first one we opened is running the server) and run the following command:

```
sudo docker-compose exec luigi python -m luigi --module youtube_data
    Elasticsearch --videoID <videoID> --index youtubeindex --doc-type
    youtubetype
```

It will run the luigi pipeline contained in youtube_data.py, using Elasticsearch as the runned task. This will trigger the rest of the tasks declared as dependencies in the Elasticsearch task The videoID is the ID of the video we want to analyse. index and doc-type set the elasticsearch parameters to store the data.

By now, we have loaded the data into the dashboard as seen in Fig. 5.2.

Now we can use the dashboard to obtain the analysed data.

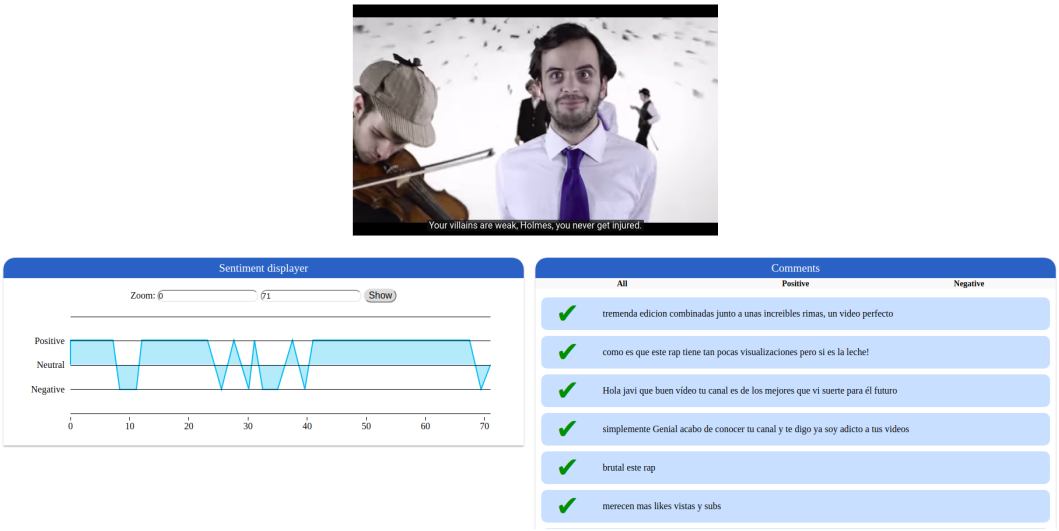

Figure 5.2: dashboard-youtube con datos

5.4 Data Visualization

First of all, we are going to start the video by clicking on it. We will see the graph displaying the data corresponding with the video time. If we click again on the video to stop it, the graph will stop as well. If we want to zoom on a certain time-lapse, we can do so by introducing the initial and final second in the text fields and clicking in the show button.

If we missed a part of the video and we want to return to that part of the analysis, we can click in any of the data points to return that second in the video. Once we play the video again, the analysed data will continue to display from that second.

If we want to see the analysed comments, we have to click one of the tree possible options:

- **All:** All comments will be shown as seen in FIG. Blue comments are positive and the red ones corresponds with the negative comments.
- **Positive:** It will display only the positive comments.
- **Negative:** It will display only the negative comments.

If we want to analyse more than one video at the same time, we can make another dashboard using the same widgets and building it. In the next chapter we'll see some possible additional uses for this project.

CHAPTER 6

Conclusions and future work

6.1 Introduction

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

6.2 Conclusions

In this project we have developed a luigi pipeline that extracts and analyses comments and captions from YouTube videos and uses elasticsearch to store this data. We've also developed a web dashboard that displays the analyzed data using three widgets. For this purpose, we used Polymer widgets and elasticsearch to build and connect the three widgets.

This project has developed:

Five luigi Tasks: Used to extract data (ExtractorTask), join and format data (FormatTask), prepare data for senpy (FetchDataTask), analyse data using Senpy (SenpyTask) and fetching data into elasticsearch (Elasticsearch)

Three widgets: yt-comments-widget, yt-sentiment-widget and yt-video-widget

Used to display analysed comments (yt-comments-widget), captions (yt-sentiment-widget) and videos (yt-video-widget)

One dashboard: dashboard-youtube Makes an integration of the whole system that allows to make conclussions about the chosen video.

In the following sections I will describe in depth the achieved goals, the problems faced and some suggestions for a future work.

6.3 Achieved goals

In the following section I will explain the achieved features and goals that are available in this project.

Extraction of Comments and Captions from a YouTube Video We achieved the extraction of this data creating a luigi task that uses youtube-dl and the youtube-API in order to obtain the desired data.

YouTube data analysis Using luigi tasks, we can fetch the data obtained from the previous tasks and analyse it attending to the sentiment that different data produces using senpy.

Display analysed data We managed to build a dashboard that uses three polymer widgets that displays different analysed data taken from elasticsearch.

6.4 Problems faced

During the development of this project we had to face some problems. These problems are listed below:

- Polymer learning: Polymer elements and the relation between them are a bit complicated when you start to use them, forcing us to pass through a long test-error period at the beginning of the project in order to make them work properly.
- Docker persistence: At the beginning of the project, the different data stored in the docker container where the widgets where running, continued showing the same data even when I changed a lot the system. This was solved by creating a script that erased every cache files created from the docker containers.

- D3.js design difficulties: The sentiment widget graph used absolute positions, so it was very difficult to align the svg graph with the rest of the widget contents and the other widgets when put into the dashboard.

6.5 Future work

In the following section I will explain the possible new features or improvements that could be done to the project.

Voice and facial expressions analysis. We focused only in text-based analysed data so, in order to improve our dashboard, we could include voice and facial expressions analysis widgets so we can obtain more information about the video.

Big data. In future projects, using the created luigi pipeline and R, we could build a system that collects data from multiple videos and make conclusions using R in order to make a more accurate vision about a creator and use it to make decisions based on this data. Luigi could be used for batch jobs in an order of thousands of them, enough to analyse a youtube creator each time. If we are more ambitious, we should use another system if we wanted to analyse all the creators in a network for example.

Bibliography

[1] EDIZIONES/Portaltic. Empresas británicas retiran su publicidad de google y youtube en protesta por los anuncios en vídeos de terroristas. *europapress. http://www.europapress.es/portaltic/internet/noticia-empresas-britanicas-retiran-publicidad-google-youtube-protesta-anuncios-videos-terroristas-20170321165904.html*, 2017.

[2] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine.* " O'Reilly Media, Inc.", 2015.

[3] mortardata. How luigi works. Available on: http://help.mortardata.com/technologies/luigi/how_luigi_works.

[4] Ignacio Corcuera Platas. Development of an emotion analysis system in a university community. Available on: https://www.gsi.dit.upm.es/index.php/es/component/jresearch/?publications_view_all=1theses_view_all=projects_view_all=0task=showview=memberid=142.

[5] Polymer Project. Polymer library feature overview. Available on: https://www.polymer-project.org/1.0/docs/devguide/feature-overview.

[6] GSI UPM. Sefarad documentation. Available on: http://sefarad.readthedocs.io/en/latest/.

[7] GSI UPM. Senpy, a sentiment and emotion analysis server in python. Available on: https://github.com/gsi-upm/senpy.

[8] GSI UPM. Senpy playground. Available on: http://senpy.cluster.gsi.dit.upm.es/.

[9] Christophe Viau. D3.js simple example. Available on: http://christopheviau.com/d3_tutorial/.

[10] YouTube. Youtube statistics. Available on: https://www.youtube.com/yt/press/en/statistics.html.

[11] Nick Qi Zhu. *Data visualization with D3. js cookbook.* Packt Publishing Ltd, 2013.