Deusto

Facultad de Ingeniería
Universidad de Deusto
Faculty of Engineering
University of Deusto

Master Universitario en Ingeniería Informática

Master's in Computer Engineering

Proyecto Fin de Máster
Master's Final Project

Brief

This project follows a architecture for providing analytical services to two different stakeholders. Offers to restaurant owners a tool for getting an analysis of the social opinion of their business. And provides to citizens a tool for letting them know about the restaurant ratings on their surroundings.

For achieving this, social networks data, an other available Open Data sources on the Web are queried. Multi-language user opinions and other unstructured data about the products and services of a restaurant business are obtained and processed through different semantic analysis techniques. The data is stored as a new graph model which is adapted to the channel of the stakeholder.

Descriptors

semantic web mining, cognitive computing, graph-oriented databases

Contents

List of Figures

Chapter A

List of Tables

Chapter 3

Chapter 4

Chapter 5

Algorithm Index

1. INTRODUCTION

Internet is evolving by leaps and bounds; the *Web 3.0* was reached more or less in the year 2012 and today people is already talking about the *Web 4.0*. To reach this new milestone, researches must help the companies to understand how to distill knowledge from unstructured or uncorrelated data. In addition, how can this knowledge can be applied to their processes to become more competitive has to be analyzed.

Nowadays, companies can explode titanic amounts of data to detect changes on people's fashion, new trends, customer behavior, etc. But these data is usually extracted from their previous experience with customers and public non-standardized sets of data.

Semantic Web Mining aims to combine the areas of Semantic Web and Web Mining for facing the challenge of examining the data available in the Web. All these data is only understandable by humans, but is so huge that it is required to use machines to take advantage of all available information. The problem is that machines cannot understand raw data because by itself has no sense. Thus, to efficiently manage all this information, a first step of data processing is required for separating the straw from the grain. During this, process the data is normalized and a semantic meaning is provided. After that, the semantic data is ready for further process it and retrieve useful information through natural language processing and other cognitive computing techniques.

1.1. SEMANTIC WEB

The *Semantic Web* is based on a vision of Tim Berners-Lee, the inventor of the World Wide Web (WWW) and is an extension of the WWW by means of new standards agreed by the World Wide Web Consortium (W3C). It provides a common framework that allows data to be shared and reused across application, enterprise and community boundaries. The main objective is allowing to machines to process web data in order to support a user when is searching data.

For instance, today is impossible - or at least very difficult - to find information through a browser just with a keyword search. Consider a site where lessons are offered. This site has stored information about its users (i.g.: its location, age, education, interests...) on one hand, and logically information about lessons that are offered properly categorized by topic. When a user is searching for a lesson, the site can apply rules for offering lessons that may be interesting for the user.

Providing semantic meaning to the web is not easy. Its structure has to be defined and standardized, a common vocabulary has to be set and a logical language has to be agreed. Finally, the structures have to be filled with data [1]. In the *Schema.org* website you can find entities that have been agreed among people for representing different kind of data.

In the figure 1.1 you can see the layers of the *Semantic Web* proposed by Tim Berners-Lee [2].

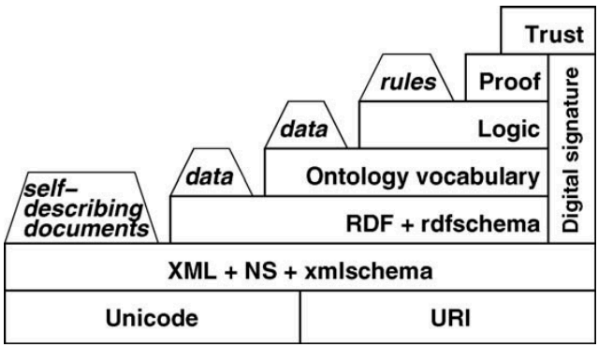

Figure 1.1: Layers of the Semantic Web

Another key concept of *Semantic Web* is that the data has to relate other data in order to make accessible all data. This is called *Linked Data*, relationships between data that makes it reachable and manageable by *Semantic Web* tools. By this way, the web of data is created instead just a collection of raw datasets with no real relationship between data [3]. Each resource is identified across all entities by means of a Unique Resource Identifier (URI).

Achieving all these points require of some standard tools and formats. Extensible Markup Language (XML) is the language used for storing data and it is extended with other frameworks:

1. XML Schema Definition (XSD): mostly used to describe property types (e.g.: integers, dates...). It can be used for establishing a set of rules that the document must fulfill.
2. Resource Description Framework (RDF): is a standard model for publishing and linking the data on the web. This framework has a feature that ease data merging and supports the evolution of schemas over time without requiring modifying all data consumers.
3. Onthology Web Language (OWL): represents rich and complex knowledge about things, group of things and relations between things. It extends the capabilities of RDF and provides a bigger vocabulary to deal with more complicated and restricted relationships [4].

1.2. WEB MINING

Web Mining consist of applying *Data Mining* techniques to the content, structure, and usage of the Web resources. Data mining is about processing data and identifying patterns and trends in that information so that you can take a decision. Mining can be applied over structured and non-structured data, which is a very useful tool for transform human readable data to machine-understandable semantics.

Normally, mining process begins at one point of a website. After that, related pages are followed to obtain more data. All gathered data from the site is processed as a whole. An important task is the evaluation process of calculate relative relevance of the pages to be analyzed, based on the hyperlink structure of the site. Data mining and the page structure mining is done at the same time most of the times.

During the latest years the *Web Mining* has become a trend. Because of that a great variety of techniques have been created to get valuable information for complex real-world problems. A lot

of scenarios can be found, from medical research to customer-business relationship management [5].

For instance, the *Data Integration and Predictive Analysis System* - funded by the *U.S. Army Medical Research and Materiel* - built a system that aimed for the prediction, analysis and response management of infectious diseases. They employed historical data, factors that could boost the incident, disease patterns, timings...along with a Support Vector Machine (SVM) for using Machine Learning to solve this problem.

Another case is detecting how vary the consumer behavior to detecting market opportunities. The solution consisted on gathering data generated by Internet users and applying text mining to derive consumer perceptions.

1.3. THE PROBLEM

Last year the city of Bilbao received a million tourists who stayed overnight in the city. So, from all tourists one million stayed in the city instead of the surrounding areas. This year the city has received the *2018 European City of the Year* award [6]. So, it can be guessed that next year the amount of tourists will increase.

When searching for information about the city, people normally searches in different web pages such as *TripAdvisor*, *Booking*, *Google Places*, etc. To perform these researching tasks can take much time, and normally there are so much reviews about a place that people only gets a scant of the reality.

Similarly, the owners of restaurants, pubs and other interest points use several social networks for advertising their business. The owners do not have enough time for managing its business and also to do marketing tasks to search for improvement points of their business. If the company is too small, they cannot afford a *Social Network Manager*.

For improving the access to information about interest points in Bilbao, different entities and projects as *WeLive* propose challenges for asking to businesses and freelance programmers solutions for this problem.

This project aims to gather all this information spread and centralize it in a single repository using *Web Mining* techniques. Then, use *Semantic processing* algorithms to process the data from all sources and generate valuable information adapted to each user. Finally, the information can be offered to the tourists and citizens by means of a map with the score and general information about the restaurant; and business owners can get the strengths and weaknesses of their business through a list of items with a score.

This project follows the steps of other similar projects that use social data and use data mining for extracting valuable information [7]. But that project focuses in extracting information about an specific domain in a general basis. While in this project domain specific and company specific information is intended to be extracted and classified.

2. SCOPE AND OBJECTIVES

The current project aim is to make use of disperse social data and open data (so called *Broad* data) to get valuable information through an analysis process on a specific domain. Then, all the knowledge that have been obtained from the data processing and information collection is displayed to the user. In this problem exist two types of clients: citizens and restaurant owners (also referred as *businessmen*).

The citizens require to know which restaurants have in their vicinities and quick quality information about them. Restaurant owners need detailed information about their business for detecting possible problems about it. The way of how the information is shown to the users is different due to the citizens just require knowing an average score without deep details. But the businessmen require detailed information on an easy-to-read format. The way of accessing to the data is also different. Citizens do not have access to sensitive data, so a public application entry point (*Public Customers*) will be available. Restaurant owners require a private application entry point (*Back-office*) for managing their restaurants, so a back-office will be provided.

Figure 2.1 represents how the system is deployed. In the leftmost side the web application retrieves data from the *Broad* data sources, this data is transformed into information through different algorithms that make use of *semantic* algorithms. Then, in the rightmost side it is represented how the citizens can access to the information through a mobile application using mobile networks. Similarly, businessman can access to their platform through the secured web portal.

There are two kinds of data that are going to be analyzed. On one side, the *general data* from a restaurant is data such as contact information, social platform scores, description, timetables…On the other side there is review data, which are comments and reviews left by the social networks users. As the information is different, each one requires a specific process for getting value from them.

These are the main challenges this project has to face:

- **Data Summarization**:
  Valuable information must be exploded from reviews and restaurant's general data using *Data Mining* and *Semantic* algorithms.
- **Request Overload**:
  The semantic analysis process is very computationally expensive, so several users using the back-office could collapse the server if they request an analysis at the same time. In addition, citizens can simultaneously request information from a mobile application.
- **User Support**:
  Both citizens and businessmen have to understand the information and be able of use it. Citizens must have a clear idea about the surrounding businesses, and businessmen must have clear the strong and weak points from their business.

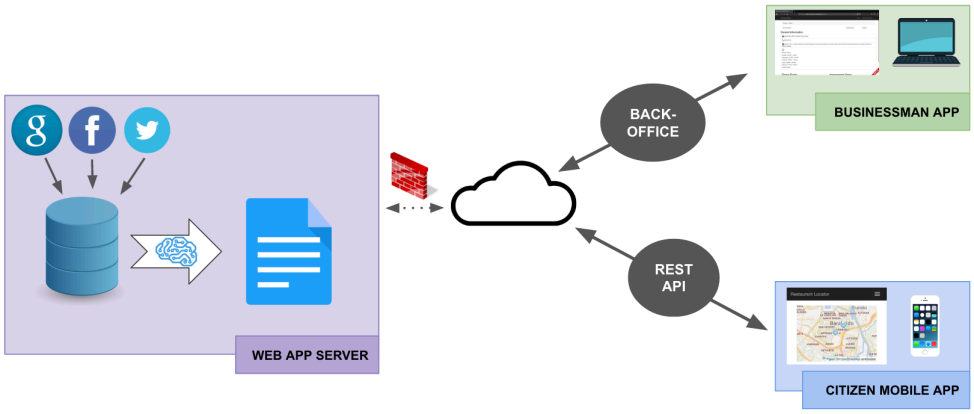

Figure 2.1: Domain Diagram of the application

- **Spam Filtering**:
  A lot of reviews from the social networks are *spam* comments, ads and other not useful data. The solution must be able to detect it and avoid entering into the analyzing phase.

This project requires knowledge about graph oriented databases for efficiently store and manage the data. Prior knowledge about Hypertext Transfer Protocol (HTTP) protocol, linked data and web engineering is required - skill with Model-View-Controller (MVC) based frameworks and web pattern knowledge. In addition, knowledge of application deployment, *GNU/Linux* based systems managing abilities are required for publishing the application in a test or/and production server. Web design basis are required for adapting the views to desktop and mobile users.

The secondary objectives of the project are the following:

- **Comparison of semantic analysis algorithms**
  Use different algorithms for analyzing the data from the user reviews using a test dataset, write the results of each review in a text file. Then, manually compare the results obtained to know which algorithm suits better in this problem.
- **Creation of *corpus* adapted to the problem's domain**
  Manually tag reviews from restaurants to get a big enough dataset for using with the semantic analysis algorithms. A specific dataset could lead to get better analysis results from other analysis.
- **Definition of a public API for developers**
  Open the processed information from the restaurants to other developers. This way, new applications can be created using this platform, what can result in retrieving new users to the service and cooperate with other companies for increasing the data quality.
- **Public demonstration in *WeLive* contest**
  Present the project advantages to the *WeLive* community, and show how this project can help: to citizens to find better places where eat, and the competitiveness of companies by detecting their strong and improvement areas.
- **App publication in a public marketplace**
  Share the application in a public marketplace in order that users can easily install and use it in the area of the Basque Country.
- **Machine Learning feed from the reviews**

The analysis tools use a tagged corpus for analyzing the data from the restaurants. Letting the user mark a review as *good* or *bad* categorized allows the user to improve the future results of the analysis.

The validation of the project will be performed with some test users that will check if the application is useful for them. In addition, the review categorization will be checked to make sure that the system provides a good ratio of properly categorized reviews. This means, that the 80% of the reviews must be correctly classified.

2.1. INTERMEDIATE PRODUCTS

There are the intermediate products that the project has generated:

- **Restaurant review demo platform**: both clients can use the platform for checking the features that future versions will implement. Basic data analysis can be performed, strong and improvement points are displayed and restaurants are marked in the *OpenGraph* map.
- **Restaurant Spanish *corpus***: a domain specific dataset for being used during data analysis.

3. PLANNING

The project estimate period is 6 months, beginning in September 2017 and ending in March 2018. In the Figures 3.1, 3.2 and 3.3 the project's network diagram is shown, the description of each Work Package is defined in the 3.1 section. An incremental development has been chosen as methodology, so that early results can be obtained from the platform, demonstrating the solution feasibility. These are the main phases of the project:

- **Task 1**: Study of the *State of the Art*.
- **Task 2**: Problem and Requirements Analysis.
- **Task 3**: Research of solution technology.
- **Task 4**: Development of preliminary solution.
- **Task 5**: Development of production solution.
- **Task 6**: Seminar sessions at *Yamada Labs*.

Event through the *Tasks 1-5* follow a chronological order, the *Task 6* is executed during all project in parallel. During the *Tasks 4 and 5* the same development strategy has been followed:

1. Design of the solution: list the requirements to be fulfilled during current phase of the project, and design the solution to be built.
2. Build of the solution: coding and integration of the solution.
3. Development of demo/showcase: create a demo for showing that the requirements are fulfilled.
4. Test Validation: validate that the platform behaves correctly and detect possible bugs of the platform.
5. Conclusions and Continuos Improvement Quality Program (CIQP): analyze the results and propose improvements for future versions.

3.1. WORK PACKAGES

In this section, the details from each Work Package is described. In the Figures 3.4, 3.5, 3.6 and 3.7 a *Gantt* diagram is shown where can be seen the preliminary schedule in green color, and the real progress of the project in red color.

Table 3.1: T1. Study of the *State of the Art*

T1. Study of the *State of the Art*	
Length	25 hours
Objectives	Research the domain of the problem and detect different ways to face the problem.
Description	• Read papers about authors that have proposed different solutions. • Look for similar solutions in the market • Search cutting-edge technologies that fit in a possible solution.

Table 3.2: T2. Problem and Requirements Analysis

T2. Problem and Requirements Analysis	
Length	25 hours
Objectives	Collect and analyze requirements. Organize functional and non-functional requirements.
Description	• Deep study of the problem. • List project's global requirements. • Divide the requirements in different iterations.

Table 3.3: T3. Research of solution technology

Research of solution technology	
Length	50 hours
Objectives	Select the technology that is going to be used in the project for solving the problem.
Description	• Study possible tools to be implemented. • Test tools and learn how to make a proper use of them. • Justify technology selection among all available tools.

Table 3.4: T4. Development of preliminary solution

Development of preliminary solution	
Length	225 hours
Objectives	Create a basic solution able to demonstrate how can be used for solving the problem.
Description	• Establish data sources • Create general architecture of the application • Try different *semantic analysis* algorithms and measure which could be better on this problem. • Deploy application to a testing infrastructure • Analyze the efficiency of the solution and study improvements

Table 3.5: T5. Development of production solution

Development of production solution	
Length	150 hours
Objectives	Adapt the basic solution generated in the *Task 4* to a production solution that fits in a real environment.
Description	• Integrate security means to the *back-office*. • Optimize solution processes and user applications. • Deploy application to a production server. • Test with real users.

Table 3.6: T6. Seminar sessions at *Yamada Labs*

Seminar sessions at *Yamada Labs*	
Length	56 hours
Objectives	Assist to seminar sessions to learn from different approaches in *Semantic Web* problems with lab mates.
Description	• Write and present a report about the month's achievements. • Assist to Lab's mates' presentations. • Debate with Lab's professors about useful techniques and algorithms in *Semantic Web* domain.

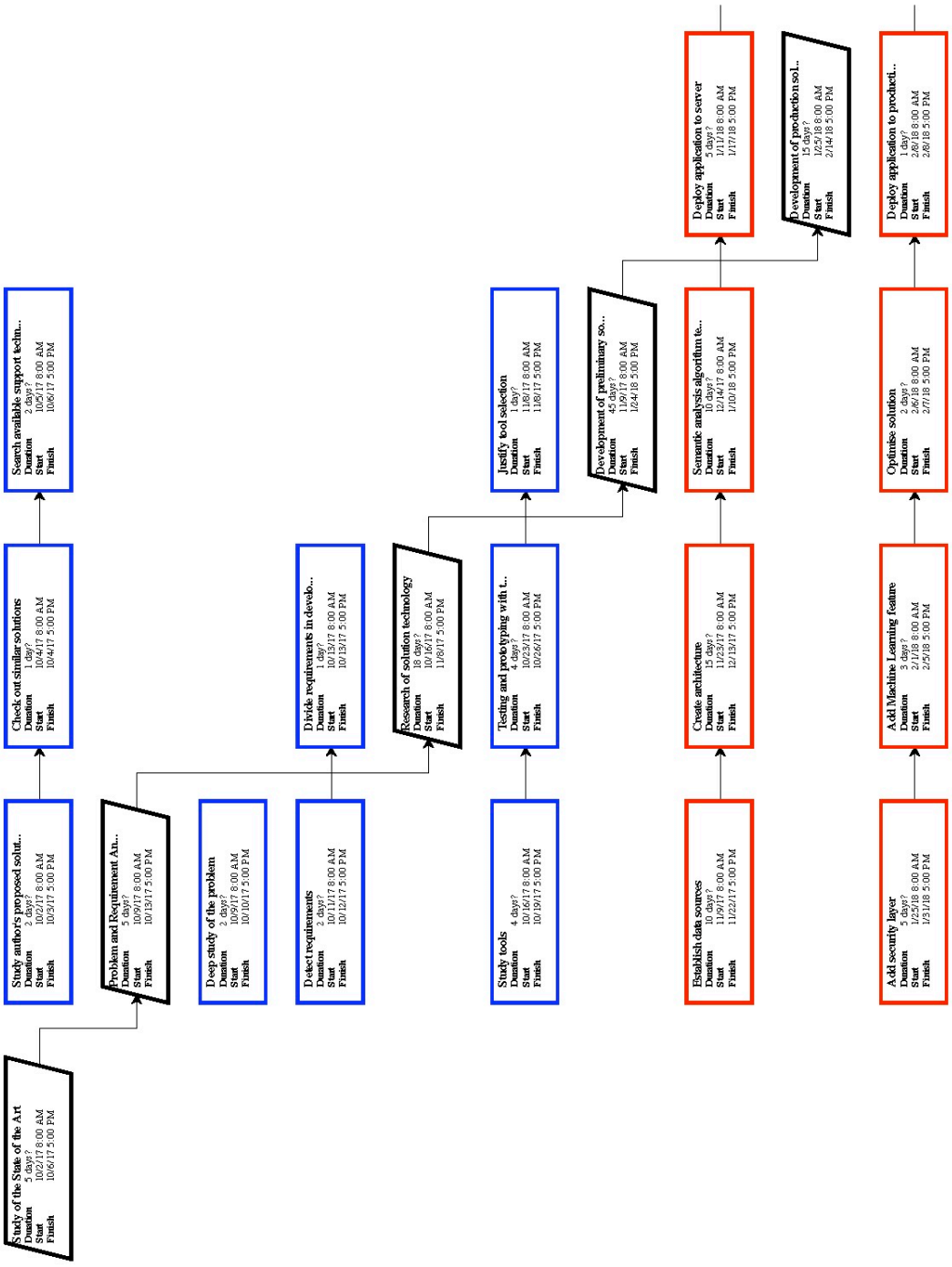

Figure 3.1: Graph Diagram (Part I)

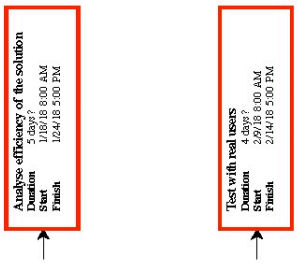

Figure 3.2: Graph Diagram (Part II)

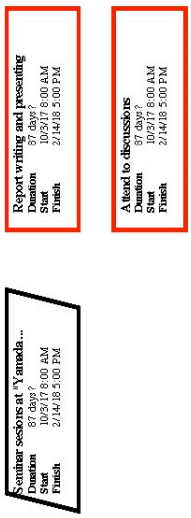

Figure 3.3: Graph Diagram (Part III)

	Name	Duration	Start	Finish	Predecessors
1	**Study of the State of the Art**	**5 days?**	**10/2/17 8:00 AM**	**10/6/17 5:00 PM**	
2	Study author's proposed solutions	2 days?	10/2/17 8:00 AM	10/3/17 5:00 PM	
3	Check out similar solutions	1 day?	10/4/17 8:00 AM	10/4/17 5:00 PM	2
4	Search available support technologies	2 days?	10/5/17 8:00 AM	10/6/17 5:00 PM	3
5	**Problem and Requirement Analysis**	**5 days?**	**10/9/17 8:00 AM**	**10/13/17 5:00 PM**	**1**
6	Deep study of the problem	2 days?	10/9/17 8:00 AM	10/10/17 5:00 PM	
7	Detect requirements	2 days?	10/11/17 8:00 AM	10/12/17 5:00 PM	
8	Divide requirements in development stages	1 day?	10/13/17 8:00 AM	10/13/17 5:00 PM	7
9	**Research of solution technology**	**18 days?**	**10/16/17 8:00 AM**	**11/8/17 5:00 PM**	**5**
10	Study tools	4 days?	10/16/17 8:00 AM	10/19/17 5:00 PM	
11	Testing and prototyping with tools	4 days?	10/23/17 8:00 AM	10/26/17 5:00 PM	10
12	Justify tool selection	1 day?	11/8/17 8:00 AM	11/8/17 5:00 PM	11
13	**Development of preliminary solution**	**45 days?**	**11/9/17 8:00 AM**	**1/24/18 5:00 PM**	**9**
14	Establish data sources	10 days?	11/9/17 8:00 AM	11/22/17 5:00 PM	
15	Create architecture	15 days?	11/23/17 8:00 AM	12/13/17 5:00 PM	14
16	Semantic analysis algorithm testing	10 days?	12/14/17 8:00 AM	1/10/18 5:00 PM	15
17	Deploy application to server	5 days?	1/11/18 8:00 AM	1/17/18 5:00 PM	16
18	Analyse efficiency of the solution	5 days?	1/18/18 8:00 AM	1/24/18 5:00 PM	17
19	**Development of production solution**	**15 days?**	**1/25/18 8:00 AM**	**2/14/18 5:00 PM**	**13**
20	Add security layer	5 days?	1/25/18 8:00 AM	1/31/18 5:00 PM	
21	Add Machine Learning feature	3 days?	2/1/18 8:00 AM	2/5/18 5:00 PM	20
22	Optimise solution	2 days?	2/6/18 8:00 AM	2/7/18 5:00 PM	21
23	Deploy application to production server	1 day?	2/8/18 8:00 AM	2/8/18 5:00 PM	22
24	Test with real users	4 days?	2/9/18 8:00 AM	2/14/18 5:00 PM	23
25	**Seminar sesions at "Yamada Labs"**	**87 days?**	**10/3/17 8:00 AM**	**2/14/18 5:00 PM**	
26	Report writing and presenting	87 days?	10/3/17 8:00 AM	2/14/18 5:00 PM	
27	Attend to discussions	87 days?	10/3/17 8:00 AM	2/14/18 5:00 PM	

Master Thesis - page1

Figure 3.4: Gantt Diagram (Part I)

Figure 3.5: Gantt Diagram (Part II)

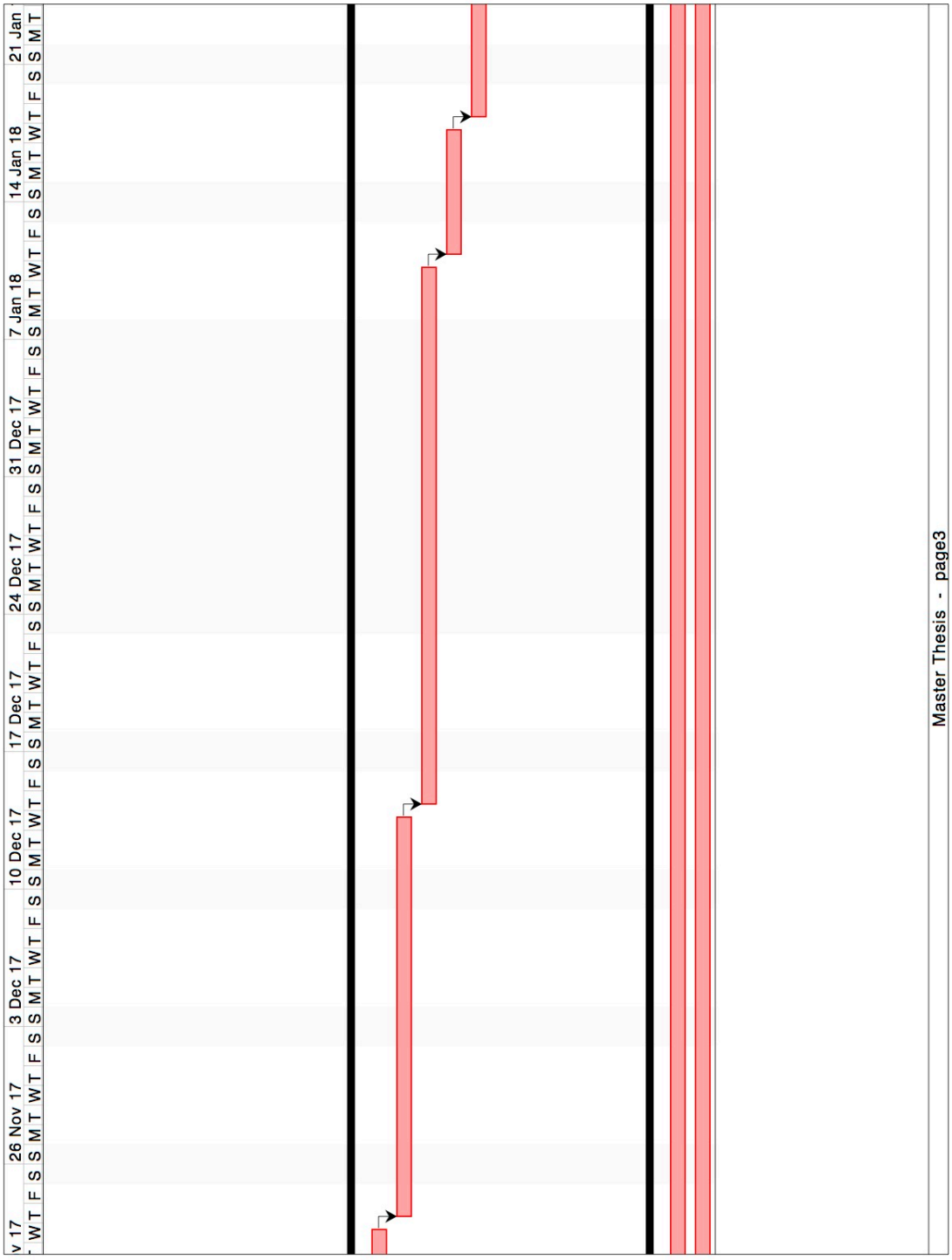

Figure 3.6: Gantt Diagram (Part III)

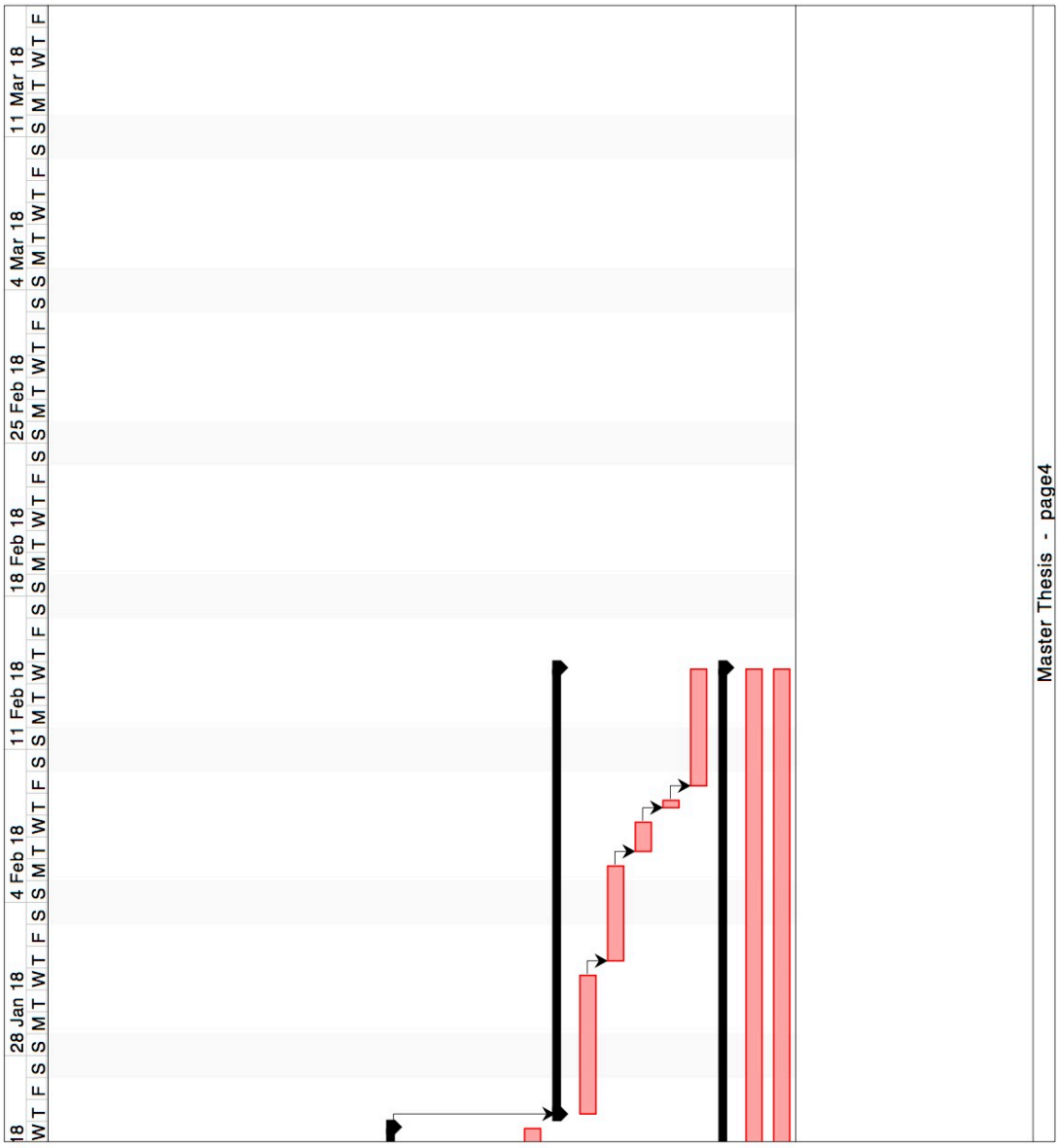

Figure 3.7: Gantt Diagram (Part IV)

4. METHODOLOGY

The project team is composed by a director from the *University of Deusto*, three professors from the *Nagaoka University of Technology* and a developer. The director from the project is also the main client from the application, because he is who decides if the application has reached its goal. The professors' advice about the analysis algorithms that can be implemented in the project. The developer has total freedom to choose the technologies which are used in the project, design, develop and deploy the application once the requirements are fulfilled. This relationship can be appreciated in the Figure 4.1.

At the beginning of the project, the director proposed the research area and a possible set of applications that were available. After discussing and choosing one of the available options, the functional requirements and the user stories were explained. Then, the developer designed a solution proposal and presented it to the director and the professors for obtaining the approval of the scheme. After that, the solution development was split into different milestones to keep an incremental track of the solution and lay out alternatives if necessary. At the Chapter 3 the details of the different parts of the project can be read.

Each month the developer has to write and defend in a seminar the latest activities performed in the project. During these seminars, all the attendants can suggest changes and give their opinion about the solution followed. At the end of a seminar, the developer can choose to add any suggestion to the list of functional or non-functional requirements of the project - depending on the requirement significance.

A milestone ends after the developed solution is checked by the director of the project. The director can suggest any change or modify the plan if necessary. When the approval is granted, the developer can add the new features to his task list. If the changes have impact in the preliminary planning, the budget and the planning have to be updated.

When the new features are developed, these have to be merged to the *stable* version of the application as soon as all tests are passed. The *stable* version can be used by a client to check out if the solution really solves the proposed problem.

4.1. EXECUTION CONDITIONS

The resources are defined in the Section 4.3. The work place is the *Yamada Lab* at the *Nagaoka University of Technology*. The laboratory is composed of three large computer rooms kitted out with cooking tools, printers, wireless and wired network connections; a meeting room with a projector; a lobby with some settees for providing a more informal environment.

The timetable and the calendar are very flexible, the student can use the lab at any time during the week, even the weekends. Although the professors are only available from Monday to Friday and

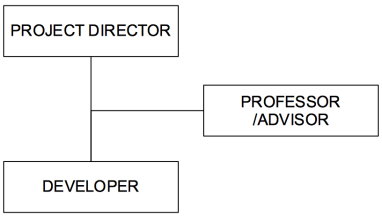

Figure 4.1: Project Org Chart

any meeting has to be previously set by e-mail. The University provides a Personal Computer, but the student can use his laptop if desired. If huge computing power is required there is a possibility of reserve the University's supercomputer to make complex calculations.

The contact between all project members is mainly through e-mail or *Skype* conference. All Requirements and decisions are stored in reports written after the meetings. All project changes have to be agreed with the project director and the developer. The final validation of the solution is performed by the project director after all components of the solution are created - once the developer has deployed the solution to the production server.

The *University of Nagaoka* is responsible of the availability of all services offered to the developer. The developer is the responsible of having a backup storage, code repository and the operation of all development systems. The main tools for documenting and storing the project are the following:

- *GitHub* is used as code repository for the product of the project.
- *Heroku* is the chosen platform as testing platform for deploying the solution.
- For keeping the track of the project's tasks *Trello* has been used.
- LaTeXand *Microsoft Office* suite have been used for writing the documentation.
- All technical diagrams have been created with *Visual Paradigm*.

4.2. PRODUCT DELIVERY

The final application, design documents, specifications and documentation have to be delivered to the project director once the milestones have been completed.

Heroku will be used as main deployment platform, for testing and product version of the product. However, another platform can be proposed as production server once a stable product is delivered. For deploying the product as stable versions, the functional requirements of the current iteration have to be accomplished and all integration have to be passed.

In case that fatal errors are found once the platform is deployed, it can be asked to roll back to the previous version to prevent a service outage. The errors detected have to be fixed as soon as possible for deploying the latest version before new changes are added to the platform.

4.3. BUDGET

All the project's expenses have been gathered in this section. The expenses reflect the cost of equipment, hired services and software for the research, development, verification and documentation of the platform.

This is a Research and Development (RD) project with no commercial purposes. Because of that, it is required to minimize all the expenses. When possible, *Open Source* tools have been used, along with some software that was available freely for University students. The expenses of live in Japan and the *Nagaoka University of Technology* are not reflected in this document.

The earning of this project depends on the social impact and the feasibility to adapt it to a commercial solution.

The expenses have been divided in three categories: equipment expenses 4.1 and work expenses 4.2. Costs with an asterisk (*) means that the tools are used in more than one project, so just a 5% of the real value is considered.

Table 4.1: Equipment Expenses

Concept	**Kind**	**Quantity**	**Unit Cost**	**Real Cost**
MacBook Pro 2015*	Hardware	1	1500€	75€
Visual Paradigm	Software	1	0€	0€
Pixelmator*	Software	1	17€	0.85€
PyCharm	Software	1	0€	0€
Microsoft Office	Software	1	0€	0€
ProjectLibre	Software	1	0€	0€
LaTeX	Software	1	0€	0€
MacOS	Software	1	0€	0€
Python *DK*	Software	1	0€	0€
Neo4j	Software	1	0€	0€
Trello	SaaS	1	0€	0€
Heroku	SaaS	1	0€	0€
GitHub Repository	SaaS	1	0€	0€
Office tools	Mixed	1	15€	15€
				90.85€

Table 4.2: Work Expenses

Concept	**Cost per hour**	**Hours**	**Total Cost**
Study of the State of the Art	15€	25	375€
Requirement analysis	35€	25	875€
Research of solution technology	15€	25	375€
Development of preliminary solution	30€	225	6.750€
Development of production solution	30€	150	€
Seminar sessions at *Yamada Labs*	28€	20	€
Generate Documentation	20€	20	400€
			90.85€

5. DEVELOPMENT

The main purpose of this project is to create a web platform solution for solving the problem explained at the 1.3 section.

This section gathers the detailed description of which technologies have been used, the stakeholders and system requirements, how the platforms have been built, tested and deployed. During the development of the solution, different machine learning techniques and algorithms have been studied.

Additionally, Section 5.8 depicts the analytical procedures test results and the procedures that have been followed for achieving them are explained.

5.1. TECHNOLOGIES

In this section which tools and technologies have been employed are listed. The list includes software solutions and SaaS tools that have been used.

5.1.1 Software

The main programming language has been *Python 3.6* because is one of the best languages for fast-prototyping applications. Also, this language is compatible with a good variety of computing science frameworks used for semantic analysis and data mining. In the *requirements.txt* file in the project the whole libraries can be found. But the following libraries are the most remarkable in the project:

- **Flask** [8]
  It is a light framework that provides the most basic tools in order to develop a web page backend. The framework's object is offering a fast response framework where the user has the capability to install just the tools the developer requires.
- **TextBlob** [9]
  It is a library for processing textual data. Provides a simple *API* for performing common natural processing tasks such as sentiment analysis, classification, language detection, word tagging...In addition, it allows to develop a language grammar for processing natively a language - by default only English language can be used for some low-level tasks as sentiment analysis.
- **Scikit-learn & Sklearn** [10]
  Provides machine learning tools for classification, regression, clustering, pre-processing, model selection and dimensionality reduction tasks. This project is a classification type problem, so just some data mining and data analysis tools are used from these libraries.

For the frontend view development *HTML 5*, *ES 2015* and *SCSS* technologies have been used. The whole list of required libraries can be found in the *packages.json* file at the *GitHub* repository. The most important tools used on the frontend development are the following:

- **Yarn** [11]
  It is a *Javascript* dependency management tool that downloads the packages from Node Package Manager (NPM)(*NodeJS* package manager). Some advantages of *Yarn* over traditional NPM are that *Yarn* caches the exact version of your packages, in order to achieve a deterministic behavior. Also, *Yarn* installs the packages in parallel, so it is faster than NPM.
- **Webpack** [12]
  It is a Web project manager. Eases the compilation and package process of the web application. You have just to set up where the entry file is, and when launching *Webpack* it follows all import declarations for including all files to the final file. In addition, allows you to split your application in different bundles, minimize code, real-time window refreshing during the development…
- **React** [13]
  It is a framework developed by *Facebook* which eases the development and reusability of web components. One of its main strengths is that when a component requires to be updated, instead of refreshing the entire page, just the component that requires to be updated is modified. This is possible due the existence of a virtual Document Object Model (DOM) in memory which keeps trace of different events in the web application.

PyCharm has been used as Integrated Development Environment (IDE) because number of tools that provide for speeding up the development process. The University of Deusto student license allows to use this application with all its features, so there is no reason why not using this application as development platform.

Neo4j is a graph oriented database. Dealing with huge amounts of data makes impossible to use classical solutions as relational databases. This is because each restaurant is going to store different kind of data: some have more details about some services that another restaurant may not have. Using a relation database could lead to huge number of fields and relationships that could only store $NULL$ values. This will cause a drop in the performance of the database, and an increase in the storage inversion. Furthermore, $JOIN$ operations of relational databases use high amounts of memory and Central Processing Unit (CPU), and do not scale well with big amounts of data [14]. In addition, this platform has also been chosen for three main reasons. The first one is that is a free project where everyone can provide improvements to the project. The second one – which is powered by the first reason – is that is supported by strong companies like *IBM*, *Amazon* or *Microsoft* [15]. These two points usually means a good community support. The final reason is that has API implementations for several languages, which allows to programmers to choose different programming tools.

For documenting the project LaTeXhas been used as text processor - along with the *Atom* text editor -, *Virtual Paradigm* for generating *UML* diagrams, *Pixelmator* for image editing and *Project Libre* for project management.

5.1.2 SaaS

GitHub has been the code repository for all products of the project. The academic license provides private repository creation at no cost. Also has a good integration for deploying the project with *Heroku*, which has been used a testing server during the project. *Heroku* is a cloud platform that allows to deploy web application. Graph database support was vital for the project, and this platform can easily be set up for using *Neo4j* for free.

5.2. REQUIREMENTS

This platform has tight requirements in terms of User Experience (UX) and application stability. Due to the complexity of the operations that have to be performed to extract data, the system may face with an important work load. Furthermore, the user has to be able to easily understand the conclusions that the system returns in two different environments (restaurant owners and citizen environment). The functional requirements specification are the following:

- The platform has to be able to queue requests to not saturate the system.
- The platform has to filter spam or reviews containing no data.
- The restaurant owners have to be able to set up his restaurant social networks and retrieve its client's reviews.
- The restaurant owners have to be able to request a data analysis.
- The restaurant owners require an easy User Interface (UI) for knowing the strong and improvement points of his business.
- The citizen must understand which restaurants has near and which suits best for him.
- The citizen must know which is his location.

There are other requirements that are not important in an early adoption of the platform but that at mid-long term have to be implemented on the platform for improving its throughput and result accuracy. The non-functional requirements are the following:

- The restaurant owners should be able to mark a review as spam.
- The restaurant owners should be able to mark a review as good or bad, if the review is wrongly categorized.
- A specific domain *data corpus* should be generated for improving efficiency.
- The system should prioritize recent reviews.
- An API for developers should be offered for sharing restaurant information.

In the Figure 5.1 the user interaction requirements are represented. The restaurant owners have to be able to create the project, set up the social network sources and request the retrieval and analysis of the restaurant data. After that, they have to be able to read the results. Citizens have to be able to know which restaurants have near, and when they select one restaurant they need to know the details of the selected restaurant. Finally, the developers should be able to access to a read only API for using the restaurant information in their applications.

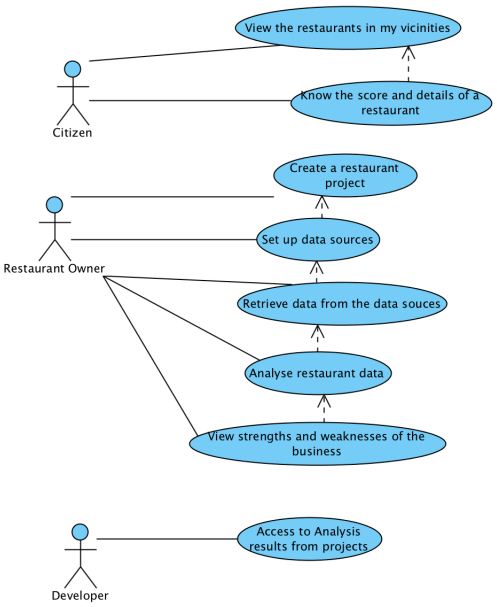

Figure 5.1: Functional requirements Use Case Diagram

5.3. SYSTEM ARCHITECTURE

As shown in the Figure 5.2 several units takes part in the whole system. Next, a brief explanation is offered, in following sections more details are given.

The main part of the solution is a web application hosted at *Heroku*, which retrieves the data from broad data sources, processes the data and returns the information to the clients requesting it. The web application is divided into several modules for ease scalability, on that way, if some module is required to be moved to another spot, the code modification will be minimum.

The graph oriented database is hosted in a different server because the *GrapheneDB* service is offered by a different company, but can be easily linked with the web application.

Twitter, *Facebook* and *Google Places* are used as social data sources. During this project, social network public data has been analyzed. But in a production environment, the clients should grant permission to the system to access the whole user reviews.

Open Data Euskadi repository is used for getting a list of the restaurants that are in the Basque Country territory - in order to detect potential clients that could use this solution. From the platform *WeLive* developers can access to a read-only API for accessing the restaurant information.

Citizens and restaurant owners access to the solution through a web browser. The web application is adapted to mobile and desktop environments, so each person can choose the platform they are more used to.

5.3.1 Solution Model

In the Figure 5.3 you can see the model followed during the implementation as a UML class diagram. The *Restaurant* and *Review* entities are based on a schema of *Schema.org* webpage – with some changes for fulfilling client requirements. Each project holds an amount of social network data

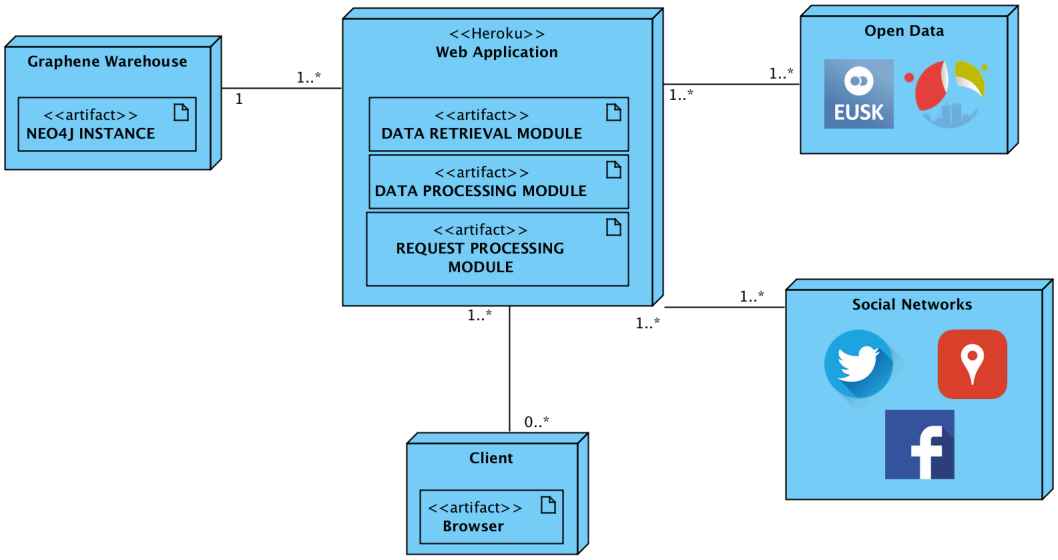

Figure 5.2: System Deployment Diagram

sources that stores all information of the restaurant available in the data source introduced by the user. In addition, all reviews that are found in the repository are stored as a *Review* instance.

In the rightmost side of the model you can see the Analysis classes. The *Analysis* class contains information about when has been performed the latest analysis, and also holds a relationship to two items:

1. An *AnalysisRestaurant* instance that holds the general data about the processed restaurants from the social networks - examples of data are the description, ranking, location...of the restaurant.
2. The *Point* class represents key points that have been extracted from the reviews along with the review the key point has been mentioned by an user and its score. The *Keyword* class represents domain keywords that want to be detected in the reviews. It is linked to other keywords that can be synonyms of a keyword because similar keywords are going to be grouped for avoiding a too long list of keywords. It is also linked to keywords in different language because of the same reason - same meaning words in different language are also grouped.

   In the Figure 5.4 a representation of the main domain objects stored as *Neo4j* nodes in the Database Management System (DBMS) is shown.

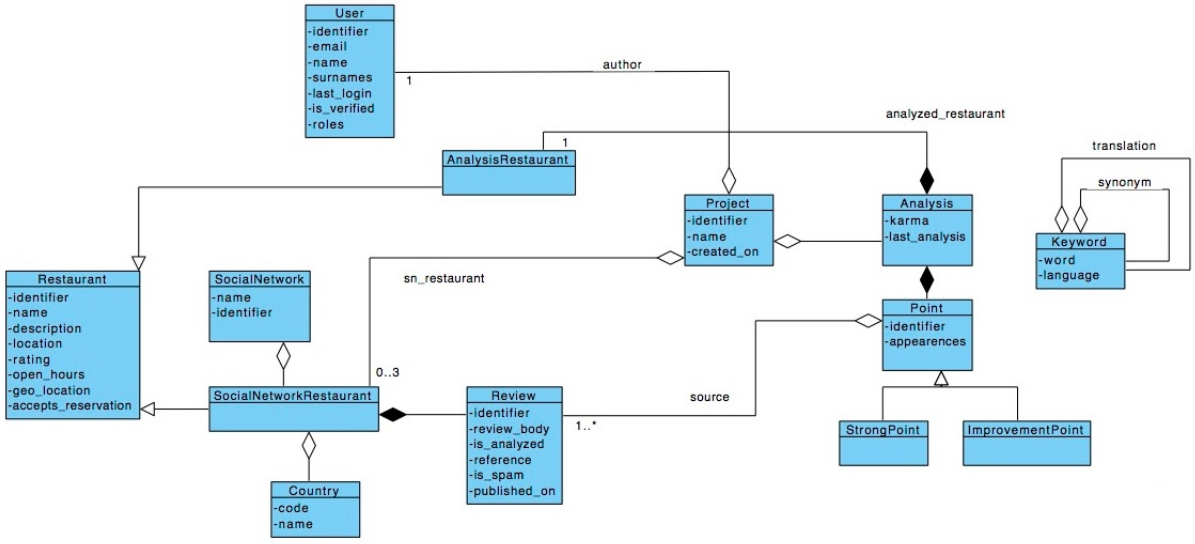

Figure 5.3: Model Class Diagram

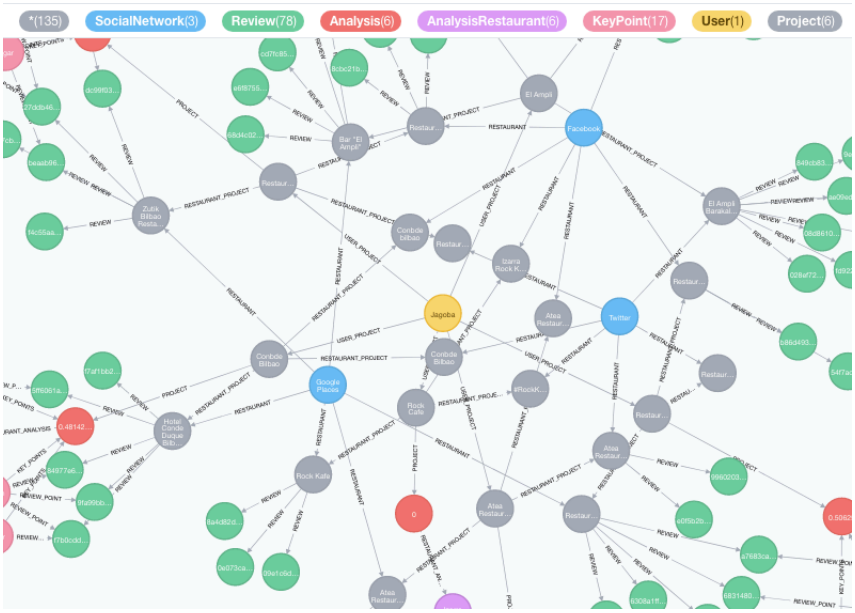

Figure 5.4: Example of main domain nodes represented in *Neo4j*

5.3.2 Design Patterns

For ease the scaling of the solution implementation several software patterns have been followed.

(a) **Repository**.
Mediates between the domain and data layers of the application. All the persistence logic is implemented in the Repository class, the rest of the application will not know anything about how the information is persisted.

(b) **Dependency Injection**:
Avoids to instantiate several times classes that just require one instance at the beginning of the application. Later, the instances can be retrieved through a dependency manager by means of an identifier. All classes that are desired to be created at startup time are defined in a XML file with the dependencies that are required for building the object.

(c) **Data Transformer**:
Transforms domain object into a plain object in order to avoid unauthorized manipulation.

(d) **Command/Query Bus**:
Allows to encapsulate application tasks into small classes with a single purpose – which makes the code much easier to read. In addition, allows to queue tasks instead of running the code synchronously. The difference between Command and Query is that the *Command Bus* objective is to persist data and that the *Query Bus* aim is to retrieve data.

5.3.3 Web Application Components

The core of the application is divided in three components that consume data from broadway data sources and retrieves the information to the platform clients. In the following subsections, the task of each component is described.

Data Retrieval Module

This module oversees accessing to the social networks that the user has set up previously and downloads the restaurant's general data and reviews. Each platform retrieves the data with a different format, so the module gets the information from the request response and stores it the database in a standardized way.
When a new information retrieval is requested, previous reviews are not removed. So, only new reviews are stored in the database.

Data Processing Module

There are two different main objectives during the semantic analysis:

(a) Decide what general information from the three sources - location, description, timetables...- is the most accurate.

(b) Analyze the reviews to detect strong and improvement points of the business.

The general information analysis process gets the stored information of the restaurant from the social networks and selects the information that is more complete. Next, the data candidate selection is explained:

(a) The application removes all the characters that may be left over (spaces, carry returns, etc.).

(b) Apply a regular expression over the data for discarding irrelevant data. Each field has an specific regular expression. For instance, a telephone number regular expression is applied to the telephone field.

(c) If two candidates are left, get the one with more characters.

In the Figure 5.5 you can see the overview of review analysis activity. Firstly, the filter of the reviews checks if the review has been previously classified as non analyzable, if the text of the reviews are not in the supported languages and if the length of the review is long enough for being analyzed; as different language review may exist, in the filtering step English and Spanish reviews are stored in two different review list.

A review is previously classified as non analyzable if it has been catalogued as spam, or if the user has manually marked that a review is spam. It is stored in the database instead of removing it for avoid downloading again and re-analyzing this data.

The platform allows to let the user mark if a review has been categorized wrongly. When a review is showed as negative and the user marks it as positive, the polarity indicated by the user replaces the previous score. Then, another filter checks if a review has already been tagged by the user. This reviews are separated from the rest, and the categorization of the user is applied to the review. Returning to the previous example, if the user has categorized the review as positive, the review will have a positive score.

After all filters, the English and Spanish sentiment analysis on the remaining reviews is performed. This process is done in parallel due to the fact that they do not depend each other. The sentiment analysis provides a review score between -1 and 1, positive if it is a positive review and negative if is a negative review. After the analysis, both language's results are mixed to calculate the overall score. The overall score is calculated thanks to the keyword list stored in the database that relates different keywords that have a synonym or translation relationship.

In the Table 5.1 you can see an output example of the results that the system returns at the end of the process. The outcome is stored in the database.

More details about the semantic analysis step are explained in the Section 5.8.

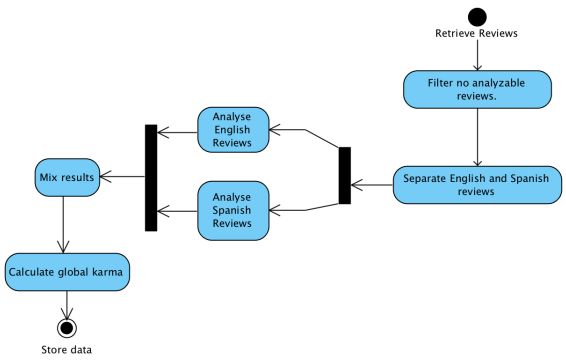

Figure 5.5: Review Analysis Activity

Table 5.1: Bilbao located McDonalds analysis results

Word	**Text Appearances**	**Karma**
Cost	1	−0.304
Meal	6	0.554
Drink	6	0.211
Offer	33	0.218
Product	3	0.537
Flavor	27	0.327
Dinner	7	0.401
Quality	1	0.5

Request Processing Module

The request processing module is implemented through the controller of the *Flask* framework. There are two possible sources to the controller:

(a) **Requests from the *WeLive* platform**
Those are requests from developers that have been registered in the *WeLive* platform and have permission to read from the server. These requests are first checked if are correctly built. If so, the desired information is retrieved, if not an error message is returned. In the Appendix A.1, the developer API manual is shown. Additionally, the manual and the end point addresses can also be found at the *WeLive* marketplace.

(b) **Requests from the Web Application**
All requests from the web application are called through *AJAX*. For security means, the origin of the requests and the headers of the requests are checked for avoiding non-unauthorized access.

5.3.4 Logging

Having as much information from how the system is working is vital for quickly response to unexpected behavior. *Python* logging system is used for generating that information and storing in *log* files.

In the development process, the *DEBUG* logging level is used in strategic places for knowing the output of the system at a point. For instance, helps to the programmer to check out if the analysis process is working correctly by recording the data structures content in some processes.

In the deployment platform several levels are normally enabled:

- ***WARNING* level**: is used for logging when a request has been invalid but that possibility was previously predicted, so a valid response was returned. Then, it is activated at the first stages of a new feature implementation for checking if the requests are correctly constructed.
- ***ERROR* level**: used for logging server errors that have been able to recover and send a response to the user. When a log with this level is detected, the incident has to be checked by the administrator. For instance, it could be that requests to a social network cannot be sent because their API has changed.
- ***CRITICAL* level**: used for logging fatal server errors. This errors lead to an

unexpected application behavior. This kind of errors have to be checked and solved urgently. Because of that urgency, if an error is dispatched at this level a message is sent to a e-mail address to the site administrator.

5.4. FRONTEND FRAMEWORK

Instead of working with an already prepared framework for develop and deployment level, as could be *Angular*, a customized framework has been built for working with all the desired libraries. Nevertheless, at client level only the *React* framework will be noticed.

Yarn provides and manages *Javascript* libraries. Also, allows to create some alias to execute some tasks. At the *package.json* file in the *web* folder the full configuration file can be found. But at the Algorithm 5.4 it can be noticed that the *heroku-prebuild*, *heroku-postbuild* and *prod* are tasks that are executed during the deployment process. *Build* task executes the *Webpack* application, which is explained below, and *start* launches the development server.

Algorithm 5.1: Extracted from the *Yarn* configuration file

```
{
  "name": "restaurant_reviews",
  "version": "1.0.0",
  "author": "Jagoba P. G.",
  "license": "MIT",
  "devDependencies": {
    // [...]
  },
  "dependencies": {
    // [...]
  },
  "scripts": {
    "heroku-prebuild": "NPM_CONFIG_PRODUCTION=false NODE_ENV=deve...",
    "heroku-postbuild": "npm run prod; export NPM_CONFIG_PRODUCTI...",
    "build": "webpack",
    "start": "webpack-dev-server --progress",
    "prod": "PROD_ENV=1 webpack"
  }
}
```

Webpack tool allows to compile all *Javascript* and files automatically. So, just the source Javascript (JS) files are linked from the *webpack.config.js* file and then the rest of dependencies are found by this tool by means of following the *import* statements. In the Algorithm 5.4 it can be read that three entry files are set up:

- The *admin* entry file is the back-office, which the restaurant owners use for managing their projects.
- The *client* entry file is the public area, which the citizens use for locating the near restaurants.
- The *authentication* entry file is the log in and register page. This area is used

for requesting the credentials to registered users, and allowing new users to register.

A *loader* in *Webpack* is a transformation that is performed to the code if the regular expression in the *test* key match. If the Algorithm 5.4 is read further, at the *loaders* key, it can be found that there are loaders for parsing JS files written in *ES 2015*, files, images and fonts.

Wrapping up, when the *Webpack* tool is launched will apply its loaders on the code that is found following the entry files. Then all the processed code will be written at the output file. There is also a minify task applied to the code for reducing the size of the files that the client will download when visiting the web page.

is used instead of raw *CSS* because of the much simpler language notation. Saves much time writing and debugging stylesheets. In addition, when the code is compiled, some properties that are sensitive to some browsers are transformed to the browser-specific properties. Variables and loops can be declared, so definitely is much easier to maintain this kind of code instead of raw *CSS*.

Algorithm 5.2: Extracted from the *Webpack* configuration file

```
// [...]
entry: {
  admin: "./web/src/admin/index.js",
  client: "./web/src/client/index.js",
  authentication: "./web/src/admin/authentication.js"
},
output: {
  filename: "[name]_bundle.js",
  path: path.resolve(__dirname, "web/static")
}
// [...]
"module": {
  "loaders": [
    {
      "test": /\.js$/,
      "exclude": /node_modules/,
      "loader": "babel-loader",
      "query": {
        "presets": ["es2015", "react"]
      }
    }, {
      "test": /\.(png|jpg|gif)$/,
      "loader": "file-loader?name=img/[name].[ext]"
    }, {
      "test": /\.(scss|css)$/,
      "use": [{
        "loader": "style-loader"
      }, {
        "loader": "css-loader"
      }, {
        "loader": "sass-loader"
      }]
    }, {
      "test": /\.(eot|woff|woff2|svg|ttf)([\?]?.*)$/,
      "loader": "file-loader"
    }]
}
// [...]
```

As *Webpack* packages everything in a single file per entry file. When linking the template with the just one file has to be referenced through the *script* tag.

As there is no professional designer in the project team, *Bootstrap* has been chosen as web-styling library. There is library adapted for running with *React*[16], which have each framework component as a class. So, when using the *Bootstrap*'s components instead of writing layers and layers of divs, classes which retrieves the code are being instanced.

As the back-office side has several views, *react-router* library is used for emulating the building. This library allows to declare the s of the application in one component joined with the component that will be rendered. This removes the need of managing which view is being rendered by hand.

For managing the state of the application, and passing the information correctly from one view to another, *Redux* library is employed. *Redux* is a predictable state container for *Javascript*. As the amount of component increases, the complexity of the application rises. To avoid side-effects when firing some action in a component, the state of the application is handled in a global container that changes as a component fires dispatches actions. A detailed document about the motivations of this popular library can be found at the project's home page [17].

5.5. USER MANUAL

Two types of clients will access to the web application. On one side, the citizens, that use a public site for checking the restaurants that have near. On the other side, the restaurant owners, who use a private site for managing the data sources of their restaurant and obtaining valuable information from the analysis process. In this section, each site manual can be found.

5.5.1 Citizen view

This site has free access to all visitors. When an user enters into the site is asked to grant permission to access to the location. This is required for locating the near restaurants and keep the trace of the user. The location is sent to the server when the user has moved significantly, for not overwhelming the server with requests.
Then, when the server receives a request, it retrieves the restaurants that are one kilometer around the user. These restaurants are represented by a marker in the map (Figure 5.6). Also, another marker represents the user position at the map.
If the user touches a restaurant marker, a modal window will appear over the map. The modal by default displays the score of four fields about the restaurant (food quality, location convenience, expense and service quality) and the average score (Figure 5.7). Those scores are generated from the semantic analysis performed to the review data.
Clicking over 'Details' information about the restaurant will be displayed in the modal (Figure 5.8). The details fields are extracted from the data downloaded from *Facebook* and *Google Places*. If there is no information about a field, a 'N/A' label will be displayed instead.

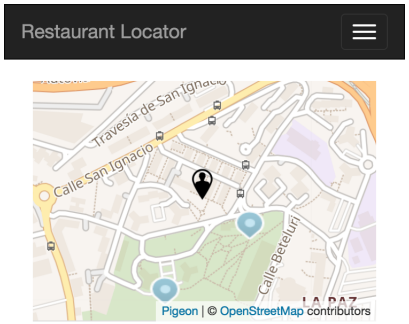

Figure 5.6: Citizen map view in mobile device

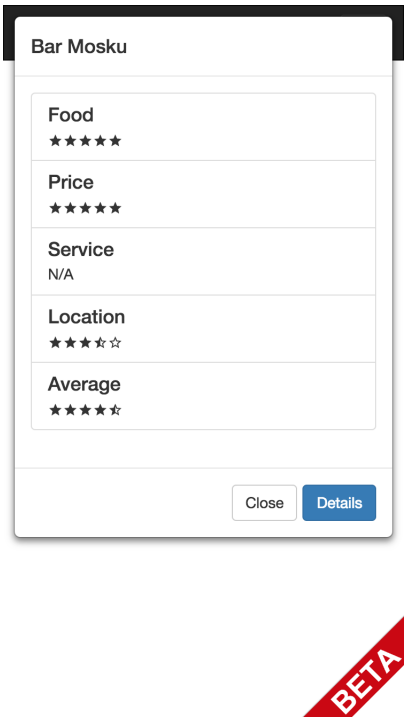

Figure 5.7: Restaurant score at citizen map

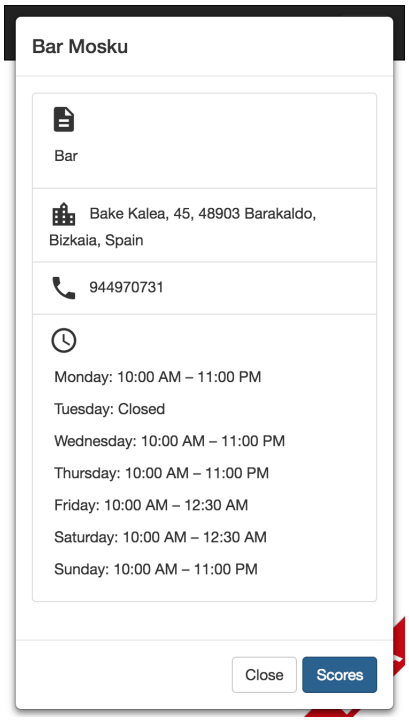

Figure 5.8: Restaurant details at citizen map

5.5.2 Restaurant owner view

This site is required previous registration for accessing (Figure 5.9). *Google* login is used for authentication, so the user is required to have a valid *Google* account. Users can freely register, but their account will not be activated until an administrator has granted them access. Due to business reasons, when an user has registered an email is sent to the administrator. So the administrator will contact with the restaurant owner to grant him access to the platform.

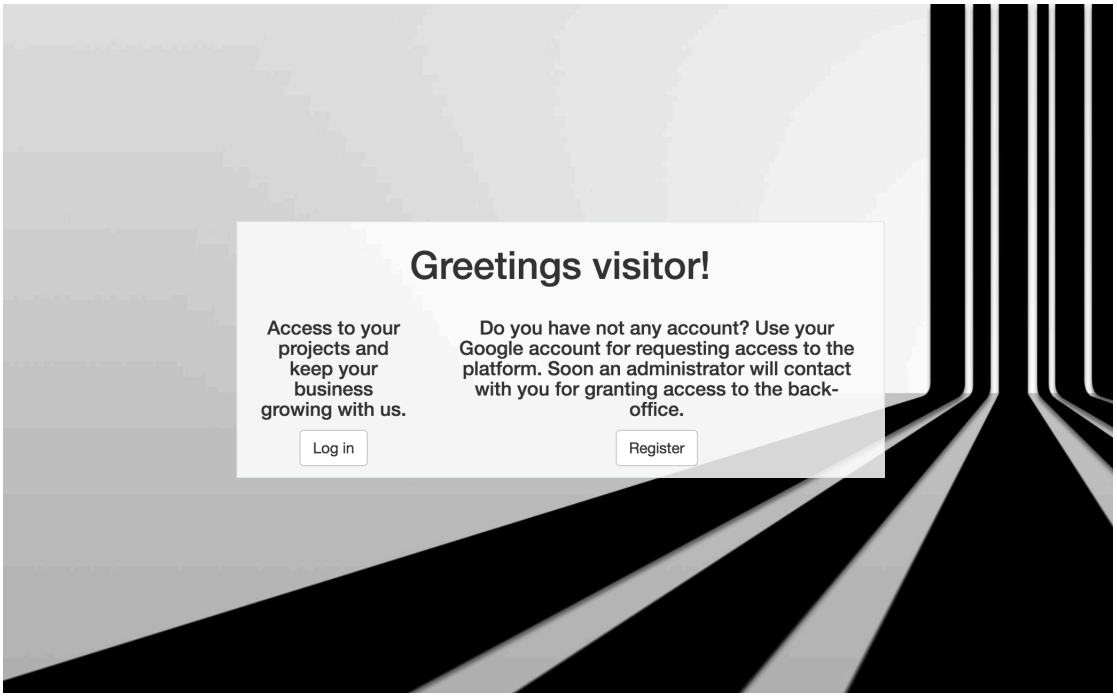

Figure 5.9: Back-office register and login form

After the user account has been activated, the user can log into the platform. In the main window, the user can view his restaurant projects. From the list the user can modify, remove the projects and see the details of the selected project. The first time the user visits the page, the list will be empty. So, the user has to create a new restaurant project by hitting the *New* button and filling the form displayed at Figure 5.10.

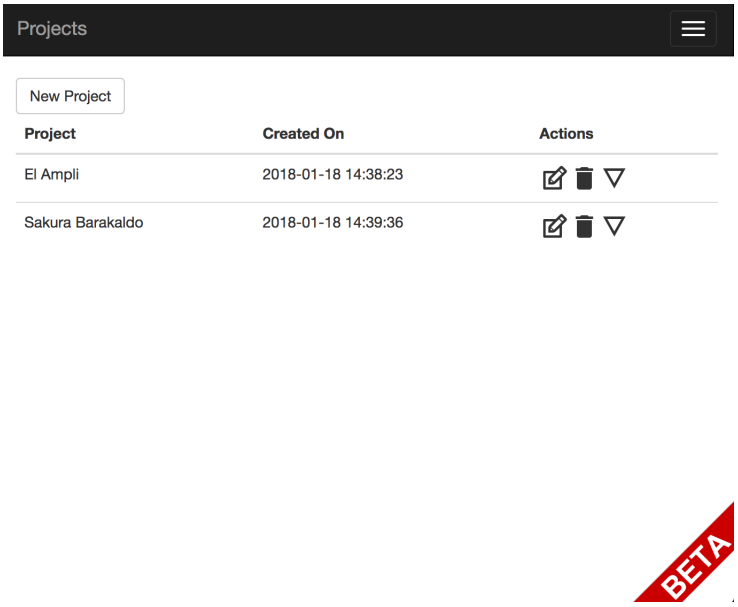

Figure 5.10: Back-office project list

Once the restaurant project has been created, the user has to set up the social network sources (Figure 5.11). When the user fill the *Facebook* or *Google Places* input fields, a valid restaurant is searching through their API. If the restaurant account is found, the name and the address of the restaurant is displayed in the list field. If several restaurant accounts are shown, the user will have to select which one belongs to him through the list box that is below the corresponding input field. The Twitter field is a different. As this social network has more flexibility when searching data, the user can make three different searches:

i. Use the character for searching tweets from a specific account.
ii. Use the # character for searching tagged tweets.
iii. Fill with raw text for searching tweets containing some text.

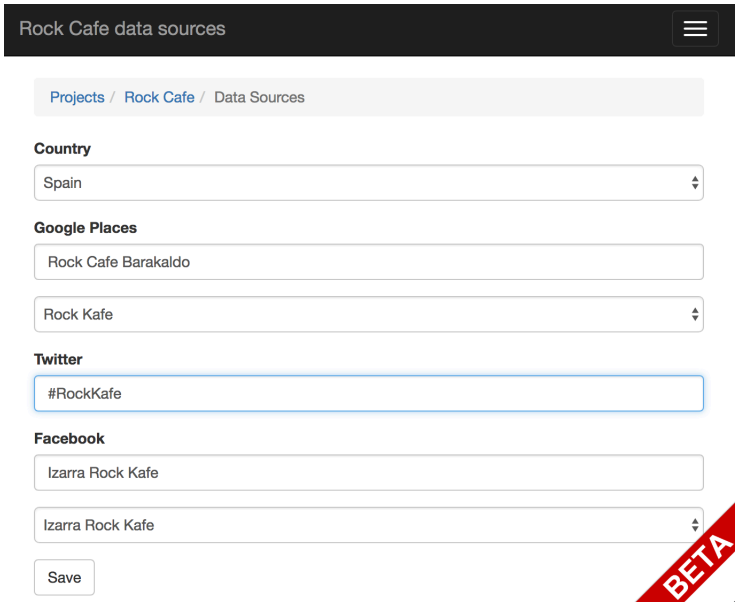

Figure 5.11: Back-office social networks set up

Once the social networks are set up, the user can request an analysis of his restaurant reviews by hitting the *Analyze* button. When an analysis is requested, the system will download all reviews from the social networks that have been set up, and then will analyze the reviews. This might take some time, depending on the amount of reviews that the restaurant has available.
While the analysis process is running a spinning roulette will be shown. If any error happens, a prompt will be shown to the user informing about the constitution of the error. If the process is successful, the page data will be reload displaying the analysis information to the user.
General restaurant information will be displayed in first place (Figure 5.12). Below, the strong and the weak points of the business will be shown (Figure 5.14) separated in two columns. Each row represent a keyword which contain linked reviews that has been analyzed, along with a tag marking if that point has an average of positive or negative reviews (*karma*).

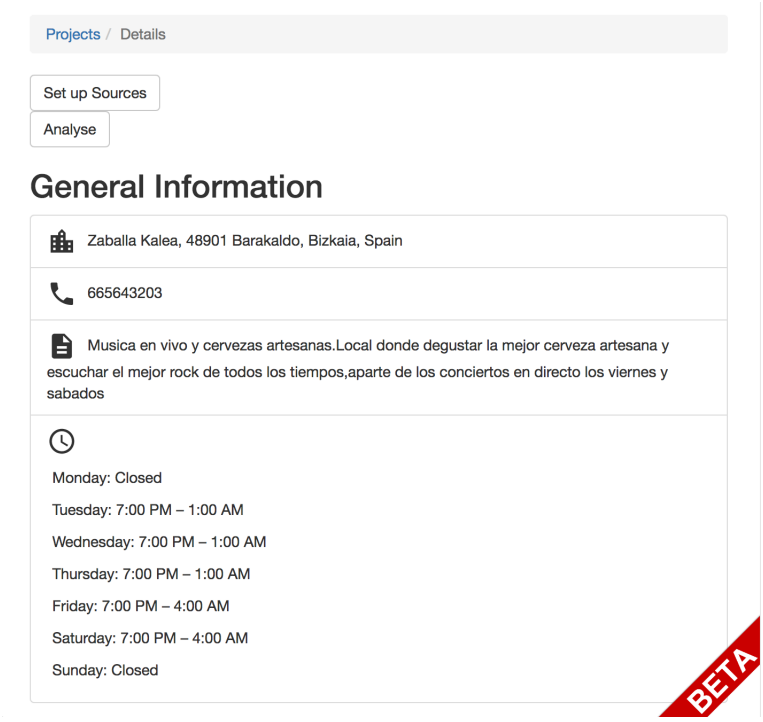

Figure 5.12: Back-office restaurant details after analysis

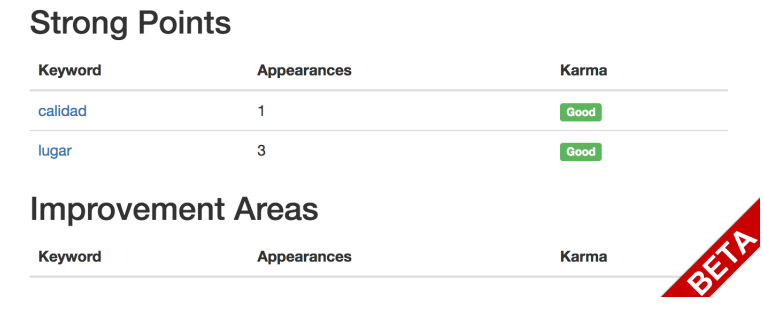

Figure 5.13: Back-office restaurant strong and weak points after analysis

When clicking over a keyword a modal window is displayed. The modal will show a list of the reviews containing the detected keyword, along with the *karma* score of that review (Figure 5.14). In this list the user will see positive and negative reviews. Remember that a keyword is classified as positive or negative depending on the **average** of all the reviews containing that word or a synonym.
If some review is spam, the user can hit *Mark as spam* link for filtering that review in a future analysis. Furthermore, if an analysis has been wrongly categorized - for instance, a positive review is marked as negative review - the user can hit the thumb up button for mark the review as positive. On this way, the next time the review is analyzed, the system will learn from the error.

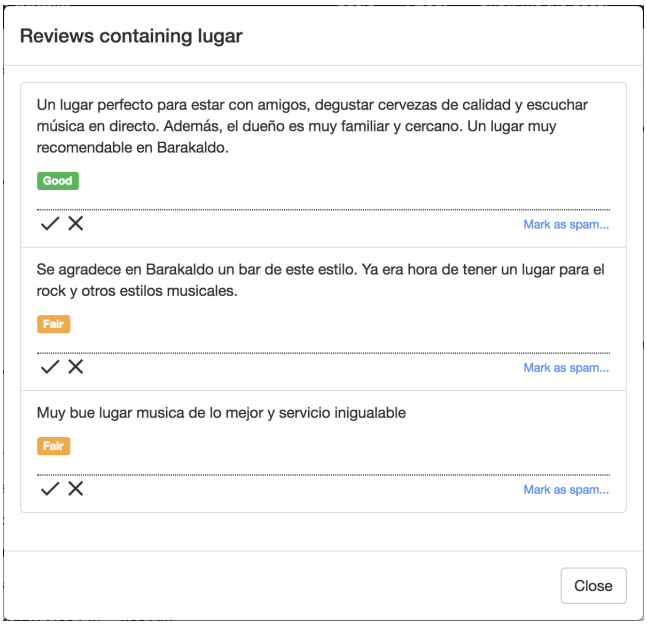

Figure 5.14: Display of reviews containing a key point

5.6. APPLICATION TESTING

For the sake of improving the user experience and the maintenance of the application, some integration test have been implemented in the backend. The tests create HTTP requests to the application end points and check if the response is the expected.
The *Python* library used for testing is called *unittest*. This library allows to the developer to create customized test cases and suites. In addition, *Coverage* library has been used for checking the amount of code that has been tested. Having a coverage test allows to detect code areas that have not been tested, or execution paths that have not been expected.
Before start writing tests, a list of the objectives that had to be reached has been performed:

- In order to detect unexpected errors when minor changes are written, tests have to be executed before deploying the platform to the production server.

- Detect lines of code or methods that are not being used and are no longer required.
- Find potential bug areas or non-secured areas.

After that, a report with the list of available end-points in the platform and a list of tests that were required in each end-point was written. This list was required for avoiding to miss some test. So, after all tests were implemented a review was done. In the Figure 5.15 a screenshot from the code coverage report is shown, after all tests are passed.
At the *README.md* file at the project's root folder, the instructions of test execution can be found.

5.7. CONTINUOUS INTEGRATION

Developed tests allow to knowing that the main features of the application do not stop working when modifying the code. Launching a test every time a push is done can be very painful though. In order to do not manually check if the new code breaks something after each push, a *Travis* hook has been set up for each time the code is pushed to *GitHub*.
Travis [18] is a distributed continuous integration service used to build and test software projects. The programmer is allowed to set up different tasks for making sure that the integrity of the new code pushed to the repository does not compromise the platform - if the tests have been properly designed.
Each time the programmer pushes the code to *GitHub*, a job is run at the *Travis* platform. The job will execute the scripts that are set up at the *.travis.yml* file. If any of the commands return a response code different to zero, means that one command has failed. When a fail happens, an e-mail is sent to the owner of the platform and the collaborators that work in the project.
Therefore, if each time a push is performed and after some minutes no alert is received to the e-mail, it means that the new code does not compromise the application. However, using this platform does not mean putting testing aside. Before pushing the code to the production platform, or after introducing a big change to the project, tests have to be executed at the developer machine.

Coverage report: 75.295%

filter...

Module ↓	statements	missing	excluded	branches	partial	coverage
src/__init__.py	0	0	0	0	0	100.000%
src/application/__init__.py	0	0	0	0	0	100.000%
src/application/analysis/__init__.py	0	0	0	0	0	100.000%
src/application/analysis/corpus_reader.py	60	31	1	26	0	45.349%
src/application/analysis/english_analysis.py	40	1	0	14	0	98.148%
src/application/analysis/rated_review_analysis.py	44	20	0	16	1	48.333%
src/application/analysis/spanish_analysis.py	101	5	0	24	2	94.400%
src/application/analysis/train_corpus.py	4	0	0	0	0	100.000%
src/application/command/__init__.py	0	0	0	0	0	100.000%
src/application/command/analysis/__init__.py	0	0	0	0	0	100.000%
src/application/command/analysis/create_analysis.py	31	1	0	0	0	96.774%
src/application/command/analysis/process_restaurant_overview.py	131	3	0	54	6	95.135%
src/application/command/analysis/process_restaurant_reviews.py	144	14	0	34	5	87.079%
src/application/command/analysis/remove_analysis.py	29	14	0	4	0	45.455%
src/application/command/country/__init__.py	0	0	0	0	0	100.000%
src/application/command/country/create_country.py	12	4	0	0	0	66.667%
src/application/command/data_source/__init__.py	0	0	0	0	0	100.000%
src/application/command/data_source/add_facebook.py	40	1	0	4	1	95.455%
src/application/command/data_source/add_google_places.py	40	2	0	4	1	88.636%
src/application/command/data_source/add_twitter.py	34	1	0	4	1	94.737%
src/application/command/data_source/edit_facebook.py	42	1	0	6	2	93.750%
src/application/command/data_source/edit_google_places.py	42	1	0	6	2	93.750%
src/application/command/data_source/edit_project_social_network_data.py	23	12	0	2	0	44.000%
src/application/command/data_source/edit_twitter.py	35	1	0	4	2	92.308%
src/application/command/data_source/retrieve_facebook.py	96	2	0	38	14	88.060%
src/application/command/data_source/retrieve_google_places.py	73	4	0	22	9	86.316%
src/application/command/data_source/retrieve_twitter.py	45	7	0	6	2	78.431%
src/application/command/keyword/__init__.py	0	0	0	0	0	100.000%
src/application/command/keyword/create_keyword.py	20	9	0	2	0	50.000%
src/application/command/project/__init__.py	0	0	0	0	0	100.000%
src/application/command/project/create_project.py	30	1	0	2	1	93.750%
src/application/command/project/delete_project.py	30	6	0	6	2	72.222%
src/application/command/project/edit_project.py	22	1	0	2	1	91.667%
src/application/command/review/__init__.py	0	0	0	0	0	100.000%
src/application/command/review/edit_sentiment.py	27	9	0	6	1	57.576%
src/application/command/review/mark_spam.py	23	6	0	4	1	66.667%
src/application/command/social_network/__init__.py	0	0	0	0	0	100.000%
src/application/command/social_network/create_social_network.py	21	11	0	2	0	43.478%
src/application/command/user/__init__.py	0	0	0	0	0	100.000%
src/application/command/user/activate_user.py	16	0	0	2	0	100.000%
src/application/command/user/create_user.py	23	12	0	2	0	44.000%
src/application/command/user/deactivate_user.py	16	0	0	2	0	100.000%
src/application/command/user/remove_user.py	16	7	0	2	0	50.000%
src/application/command/user/update_role.py	29	0	0	8	0	100.000%
src/application/config.py	13	0	0	0	0	100.000%
src/application/data_transformer/__init__.py	0	0	0	0	0	100.000%
src/application/data_transformer/analysis.py	25	18	0	6	0	22.581%
src/application/data_transformer/country.py	4	0	0	0	0	100.000%
src/application/data_transformer/project.py	12	1	0	4	2	81.250%
src/application/data_transformer/restaurant_analysis.py	4	1	0	0	0	75.000%
src/application/data_transformer/user.py	7	0	0	2	1	88.889%
src/application/fixtures/__init__.py	0	0	0	0	0	100.000%
src/application/fixtures/basic_structure_loader.py	55	21	0	8	3	58.730%
src/application/fixtures/keyword_loader.py	88	68	0	30	0	16.949%
src/application/query/analysis/__init__.py	0	0	0	0	0	100.000%
src/application/query/analysis/get_analysis_of_id.py	14	6	0	2	0	50.000%
src/application/query/analysis/get_analysis_of_project.py	18	4	0	4	2	72.727%
src/application/query/analysis/get_restaurant_data_from_analysis.py	29	17	0	6	0	34.286%
src/application/query/country/__init__.py	0	0	0	0	0	100.000%
src/application/query/country/get_countries.py	13	0	0	2	0	100.000%
src/application/query/data_sources/__init__.py	0	0	0	0	0	100.000%
src/application/query/data_sources/retrieve_facebook_id.py	30	18	0	8	0	31.579%
src/application/query/data_sources/retrieve_google_places_id.py	24	13	0	6	0	36.667%
src/application/query/project/__init__.py	0	0	0	0	0	100.000%
src/application/query/project/get_nearby_restaurants.py	38	3	0	10	3	87.500%
src/application/query/project/get_project_of_id.py	15	6	0	2	0	52.941%
src/application/query/project/get_project_of_name.py	15	7	0	2	0	47.059%
src/application/query/project/get_project_sources.py	29	1	0	6	1	94.286%
src/application/query/project/get_projects.py	13	0	0	2	0	100.000%
src/application/query/project/get_user_projects.py	24	13	0	4	0	39.286%
src/application/query/user/__init__.py	0	0	0	0	0	100.000%
src/application/query/user/check_api_key.py	12	4	0	2	0	57.143%
src/application/query/user/get_user.py	27	2	0	4	2	87.097%
src/application/query/user/get_user_role.py	20	2	0	4	2	83.333%
src/application/query/user/get_users.py	13	0	0	2	0	100.000%
src/application/query/user/has_permission_to_project.py	19	0	0	4	0	100.000%
src/domain/__init__.py	0	0	0	0	0	100.000%
src/domain/exception.py	19	0	0	0	0	100.000%
src/domain/model.py	160	17	0	20	5	85.556%
src/domain/repository.py	45	0	30	0	0	100.000%
src/infrastructure/__init__.py	0	0	0	0	0	100.000%
src/infrastructure/command_bus.py	21	2	1	4	2	84.000%
src/infrastructure/data_transformer.py	6	0	1	0	0	100.000%
src/infrastructure/dependency_injector.py	102	12	0	38	7	85.000%
src/infrastructure/query_bus.py	21	2	0	4	2	84.000%
src/persistence/__init__.py	0	0	0	0	0	100.000%
src/persistence/database/__init__.py	0	0	0	0	0	100.000%
src/persistence/database/corpus_manager.py	24	2	0	2	1	88.462%
src/persistence/database/neo4j_repository.py	121	21	0	24	5	77.931%
src/social_networks/__init__.py	0	0	0	0	0	100.000%
src/social_networks/facebook.py	56	20	0	16	4	55.556%
src/social_networks/google_places.py	16	3	0	0	0	81.250%
src/social_networks/twitter_cli.py	37	9	0	8	3	68.889%
tests.py	80	2	6	0	0	97.500%
web/__init__.py	0	0	0	0	0	100.000%
web/core.py	72	8	0	2	1	87.838%
web/routing/__init__.py	0	0	0	0	0	100.000%
web/routing/admin.py	221	32	0	36	8	83.658%
web/routing/api.py	108	64	0	8	0	37.931%
web/routing/client.py	4	0	0	0	0	100.000%
web/routing/decorator/__init__.py	0	0	0	0	0	100.000%
web/routing/decorator/decorators.py	105	20	0	26	4	80.153%
web/routing/operations/__init__.py	0	0	0	0	0	100.000%
web/routing/operations/edit_operations.py	69	12	0	12	2	80.247%
web/routing/user_authentication.py	79	62	0	8	0	19.540%
Total	**3266**	**680**	**39**	**636**	**114**	**75.295%**

coverage.py v4.4.2, created at 2018-01-19 17:45

Figure 5.15: Coverage test report

5.8. SEMANTIC ANALYSIS

The semantic analysis process explained on this section consist on predicting the score of each review, depending on whether is positive or negative. This score is called *karma* and has a value with a range of $[-1, 1]$ that measures the polarity of the review. If the review has a low value is a negative review, and if has a high value is a positive review. There are two kinds of *karma*: one referring to the general score of the restaurant, and another one referring to single nouns or concepts. Also consider that the application includes a word list containing problem domain's words, and another list containing words that are synonyms. For instance, *quality* is a word that would be desirable to detect and show in a final report attached with the *karma*.
In first place, empty reviews or not containing information reviews are discarded.
Mainly, the project requires that the module analyze English and Spanish reviews. So first it is required to detect the language to select a *corpus*. Then, the tasks of word splitting, tokenization, word polarity detection and phrase processing can be executed.
One of the tools that looked better is an European project called *OpenER* [19] which offers all tools for performing the required tasks. Each tool is focused in a single task, so can get rid of the operations that you do not want to execute. For instance, you have a tool for language detection, another one for tagging words...Then you communicate each tool with other through pipes in a shell. On this basis, you can use *Apache Spark* for speeding up processing of large datasets with no need of rewriting the application with some parallel processing libraries. Sadly, the project is not longer active, so some tools are not updated and some of the final processes are broken.
TextBlob supports nearly the same features that *OpenER* framework, with the difference that it is not modular. Other disadvantage is that has no Spanish language definition. During the first stage of the development the Spanish text were translated to English through a call to *Google API*. But the results were not good enough, so this feature is only applied to English texts.

5.8.1 English Review Analysis

As introduced, English review sentiment calculation is done using *TextBlob* libraries. At the Algorithm 5.8.1 you can read how each review is processed. For each phrase in the review the polarity is calculated. In addition, the main nouns of the text are extracted and compared with a domain word list for knowing if the review contains relevant information.

Algorithm 5.3: Steps for review analysis

```
def analyze_review(review):
  body = review.body
  for sentence in review.body:
    sentence_polarity = sentence.sentiment.polarity
    karma = calculate_average(self.karma, sentence_polarity)
    for word in sentence.split(" "):
      self.process_word(word, sentence_polarity, review.identifier)

def process_word(self, word, sentence_polarity, review_id):
  if not is_keyword(word):
    return None
  karma_average = self.found_words[word]["karma"]
  appearances = self.found_words[word]["appearances"]
  self.found_words[word]["karma"] = calculate_average(
    sentence_polarity, karma_average
  )
  self.found_words[word]["appearances"] += 1
  self.found_words[word]["reviews"].append(review_id)
```

5.8.2 Spanish Review Analysis

Spanish Society of Natural Language Processing (SSNLP) is an scientific association that promotes activities related with the study of the natural language processing [20][21] [22][23]. Access permission to some Spanish *corpus* was requested for study if this method was more effective than translating text and using a native sentiment processor. The provided *corpus* contains tagged tweets from different domains. The tags provide information about which words provide sentiment value and the class the belong to - with five possible values (very negative, negative, neutral, positive and very positive). This *corpus* has been replaced in following versions by specific domain corpora generated reviewing by hand restaurant reviews. However, the same format as the *corpus* provided by the SSNLP has been followed.

For improving the results, the system retrieves from the database reviews already reviewed by the user for extending the data provided by the *corpora*. As explained in Section 5.5, the restaurant owner can tag a review as positive or negative if the review has not been correctly classified. This direct input can improve the accuracy of the analysis because more data can be used as training set.

SVM was employed from the *Python*'s *sklearn* library to predict the sentiment of the phrases. The next process is followed for Spanish review analysis:

i. Set up Spanish stop words and punctuation characters.
ii. Train SVM with the data from the corpora and already classified reviews.
iii. For each phrase:
   A. **Tokenize phrase**: in the Algorithm 5.8.1 it is explained which are the steps followed for preparing a review to be analyzed by the SVM.
   B. **Count words**: once all words are in a matrix, detect the number of appearances of each word.

C. **Predict**: use Logistic Regression classification algorithm for predicting the phrase sentiment.

Algorithm 5.4: Spanish review analysis using Spanish *corpora*

```
def tokenize_phrase(review):
  phrase = remove_non_stop_words(review)
  tokens = tokenize(phrase)
  return stem(tokens)
```

5.8.3 Evaluation of *corpus*

SSNLP provides *corpus* for several domains (e.g.: politics, sports, latest news…). This reference data has been evaluated with a custom *corpus* generated by hand with restaurant reviews for knowing if specific sample data could improve the accuracy of the system.

An amount of $1,000$ positive and negative restaurant reviews have been gathered for testing both *corpus*. Two classification algorithms have been used: *Linear SVC* and *Gaussian SVC*. The results, as shown in the Table 5.2, suggest that the custom *corpus* provide better results than general tagged data.

Table 5.2: Evaluation of *corpus*

Title of *corpus*	**Function**	**Accuracy**
SSNLP *corpus*	Linear	0.42
SSNLP *corpus*	Gaussian	0.27
Custom *corpus*	Linear	0.55
Custom *corpus*	Gaussian	0.43

5.8.4 Evaluation of classification algorithm

Sklearn libraries provide several algorithms for classification problems. The next step has been to find which algorithm provides better results at this point. The same set of tagged data has been provided to each algorithm to test which one has as better accuracy score and speed.

Each algorithm has been executed 5 times using as training set the 20% of the custom *corpus* data and evaluating the remaining 80% left of the *corpus* data. *Timeit* library has been used for measuring the execution time of the algorithms and *sklearn.metrics* library to calculate the algorithm accuracy.

The outcome, shown in the Table 5.3, suggests that the Logistic Regression algorithm provides the best results, and also is the second fastest algorithm. So, in the final implementation of the system Logistic Regression will be chosen as evaluation algorithm for this problem.

Table 5.3: Evaluation of classification algorithm

Function	**Accuracy**	**Time (seconds)**
Linear	0.55	3.363
Gaussian Naive Bayes (NB)	0.43	3.129
Epsilon-Support Vector Regression (SVR)	0.26	3.147
Logistic Regression	0.57	3.084
K Neighbors Classifier	0.35	3.068
Decision Tree Classifier	0.43	3.112

5.9. DEPLOYMENT

For financial and technical reasons *Heroku* has been chosen as testing and production server. This platform is very versatile, because it can be easily set up with different storing tools, or set up add-ons for mailing or *CRON* tasks. Scaling and monitoring tools that are provided with the first payment option. As application is in development and requires a high computing power for the analysis, a monitoring tool is very appreciated. In the Figures 5.16 and 5.17 it can be seen some of the tools that ease the maintenance of the platform:

- The logs that are recorded when some error is thrown in the application, can be detected through the event manager. In addition, you can set up an alert when a specific log is detected.
- Memory, throughput, response time and load graphics can be used for knowing in which moment is required to scale the application with more *dynos*, or a more powerful service.
- Raw logs can be read from its own panel. When dealing with several instances of the application, *Heroku* automatically writes each process' logs in a different view.

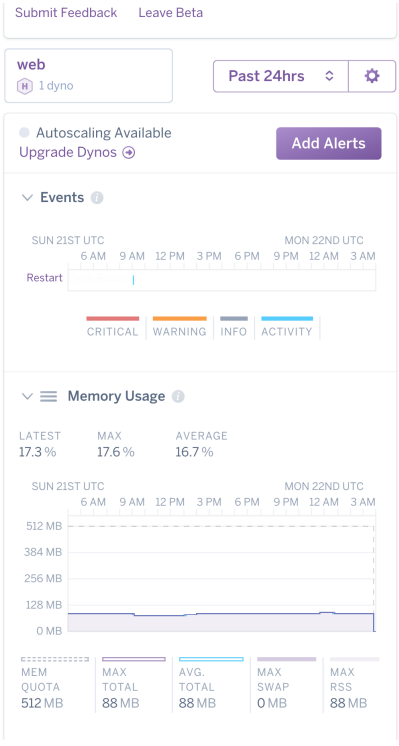

Figure 5.16: Heroku monitor system (I)

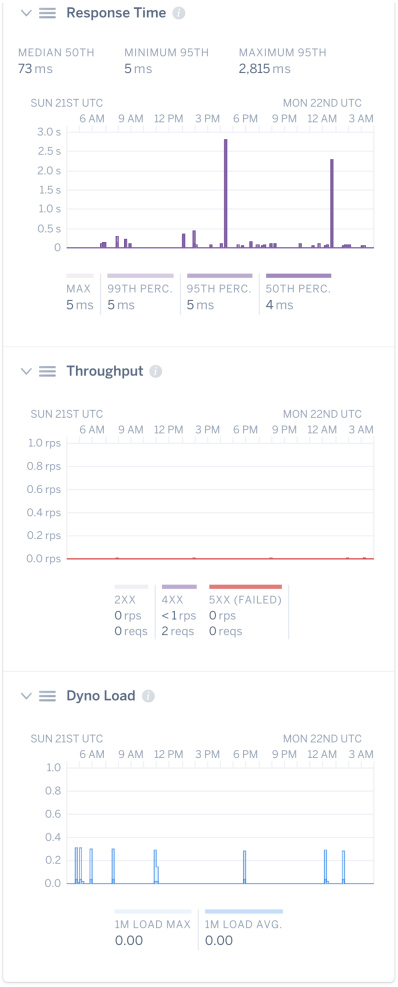

Figure 5.17: Heroku monitor system (II)

The database availability was the main reason for choosing *Heroku*. *Graphene* database provides *Neo4j* DBMS, which is the solution chosen for storing data. As free plan, the platform provides 500 MB of storage, 1,000 nodes and 10,000 relationships. The plan can be easily modified from the *Heroku* dashboard, so in case the limit is reached the system administrator can change the plan quickly.

As the application is quite complex, and several things have to be checked before deploy it, a script has been written for ease the task (Algorithm 5.9). Firstly, the *Python* environment and the application's environment variables are loaded. Then, the tests are executed. If there is any test that does not pass, the deployment is immediately stopped. If the tests are correct, the deployment begins.

Heroku allows to set up hooks for executing tasks before and after the deployment, which have been used for correctly deploy the application. Normally, the platform provides you a set of tools called *build-packs* which contain the instructions for deploying an application on a certain language or framework. This application contain two build-packs:

i. A *Node* build-pack, because the front-end has a *React* app, which require several packages to be installed.
ii. A *Python* build-pack for launching the *Flask* application.

The deployment of the application will behave as follows:

i. Deploy NPM app:
   A. Set the NPM production environment variable to $false$.
   B. Install the required JS libraries.
   C. Build the front-end application with *Webpack*.
   D. Set the NPM production environment variable to $true$.
ii. Deploy application.
   A. Download libraries indicated at *requirements.txt* file.
   B. Download *corpus* indicated at *nltk.txt*.
   C. Install *corpus* and remove source files.
   D. Run the application with *gunicorn*.

Algorithm 5.5: Deployment Script

```
#!/bin/sh -e
# Prepare environment
source venv/bin/activate
source bin/app_consts.sh

# Execute tests
python tests.py -v

# Deploy Heroku
git push heroku master
```

5.10. RELATED WORK

During the development of this project, other content has been generated for several purposes.

5.10.1 Mobility program in Nagaoka University of Technology

In first place, as part of the mobility program, several reports have been generated and presented in the seminars held at the *Yamada Lab (University of Nagaoka)*. A report has to be submitted each month, in total 4 seminars were performed. In these reports, the evolution of the platform can be followed:
In the first seminar, the project proposal, along with the technologies that were desired to be used and a general idea of the platform was presented.
In the second seminar, the first implementation of semantic analysis using *NLTK* library was presented. During this seminar, it was demonstrated that this library was not efficient enough for analyzing Spanish reviews.
In the third seminar, an alpha version of the platform was presented. The platform showed the public area of the application, where the citizens can find the near restaurants, and the private area of the application, where the restaurant owners can oversee the reviews of their social networks. The key point of this seminar was to show the Spanish review analysis using a general-purpose *corpora* and show that improvements were reached.
In the fourth seminar, the beta version of the platform was presented. The platform had most of the functional and non-functional requirements already implemented. The main part of the session was to show the different algorithms that were tested for finding out which one provided better results to the review analysis. In addition, the domain specific *corpora* was also presented, showing an improvement of the outcomes from the previous results.

5.10.2 WeLive application Contest

The platform *WeLive*[24] is a European project in which all kind of people publish problems that require to be solved through a technological implementation. For instance, a problem could be that a city requires some smart way of notifying to the city hall about a infrastructure flaw. Then, developers can propose a solution that fits the problem and develop it. The platform offers open datasets and APIs for writing all kind of applications. In addition, offers tools for defining your own API or publishing at no cost an application.
The contest consisted in developing a technological solution for an already existing problem in the platform, following some steps called *co-creation process*. The service co-creation in *WeLive* is a process that allows actors involved (citizens, companies, academies and municipalities) to collaborate with each other to improve their city by creating together useful public services, data and open processes.
As the solution proposed on this project fits perfectly with a challenge of the platform called “Interest based apps for citizens and agents of Bilbao city”, the required parti-

cipation steps were followed. A new API was created, offering the processed restaurant information. Furthermore, the application was published in *WeLive*'s platform marketplace.The result was a second position at "Best application" area.

6. CONCLUSIONS AND FUTURE WORK

In this project a complete architecture based platform, for two kind of stakeholders, has been developed. Its value has been demonstrated with real users by deploying it to a production server. In this section, a summary of all the achievements of this project are gathered, along with the conclusions extracted from the experience acquired during all project stages.

The main and secondary objectives proposed have been accomplished. Above everything, the challenges produced by the machine learning system have been hard to face.

i. The obtained user opinions were in different languages (English and Spanish). Language specific machine learning algorithms have been developed to analyze each set of reviews. *NLTK* has been used for analyzing English reviews, and a SVM has been set up for Spanish reviews. In both cases, a specific process has been develop for extracting valuable information.
ii. Lack of data has obstruct the data gathering for the early platform. This fact reduces the accuracy of the platform. But the experience on the final platform, predicts that as long as the sample data increases, the accuracy will improve.
iii. Long sentences have been to be split into smaller blocks for easing the analysis. The whole review score is not affected due to a subsequent average over each sentence score.
iv. Parallel computing libraries have been used for reducing the response time to the client.

The usage of trending technologies for platform development guarantees that the platform is easily scalable in both coding and server throughput. JS libraries reduce the amount of code that is required for managing multiple views, requirements variations and possible new features. Integration with *Heroku* platform eases the process of increasing the computing power for handling more client requests.

Another success has been to summarize multiple source unstructured data in a way that is easy for an user to understand it. The greatest value of this platform is to sum up all data in a few indicators in a map or a list. A future improvement of this point could be make a more deep split and detection of terms, for instance, score specific meals.

A great obstacle on this kind of projects is the gathering of data. This is caused by the existence of a limit on all social networks to the developer's accounts that restricts the reviews they can download. This opens the possibility to parallel projects that use some web crawlers for gathering public user's data.

During the project, several data mining tools have been tried for dealing with this problem. The results at the initial stages suggest that there is no solution for all problems. For each use case a concrete tool has to be adapted for obtaining decent results.

Some challenges that this kind of projects will face is the management of data quality.

Malicious users can feed the system with bad data. Some future lines of research are to detect these efforts to corrupt the platform. In addition, there is a lot of work to do with the context analysis and the correction of input data.

In summary, this project applies not only knowledge acquired in the Master. Software patterns and architectures, data modeling, concepts of …But also some new algorithms have been developed, a research process has been followed, and a placement in Japan where an adaptation of your project has been submitted. Finally, the project has been deployed and tested in a production environment, available for any user.

A. APPENDIX

A.1. API

The API provides information about restaurant that have been processed by semantic algorithms. Data from several sources is gathered, then strong and weak points from the restaurant is extracted from the user reviews. In addition, information about the location and timetables of the restaurant is also retrieved.
The API provides information about restaurant that have been processed by semantic algorithms. Data from several sources is gathered, then strong and weak points from the restaurant is extracted from the user reviews. In addition, information about the location and timetables of the restaurant is also retrieved. The application has a back-office where the restaurants that want to be displayed are set up. After the data sources are linked, the reviews from the social networks is retrieved. Finally, the semantic analysis can be executed and the results displayed. In the user area, a map is displayed where the near restaurants are pointed out to the user. If the user touches the restaurant signal, all the information is displayed to him.

Figure A.1: API Appendix Page 1

GET List of Projects

`/api/projects/`

DESCRIPTION

Get a list of all available projects with its identifiers.

PARAMETERS

None

RESPONSE EXAMPLE


```
[
  {
    "identifier": "e6b4577d-b80d-4819-a468-178167786140",
    "name": "El Ampli",
    "description": "",
    "created_on": "2017-11-19 13:41:29",
    "analysis_id": "9a33efa3-f199-4d22-b9a5-e5f857ca1c62"
  },
  {
    "identifier": "f78f16d4-d486-426f-8b0f-ba8751b3edaa",
    "name": "Sakura Barakaldo",
    "description": "Restaurante Japon\\u00e9s",
    "created_on": "2017-11-19 14:01:03",
    "analysis_id": "0d74639f-d008-45f3-a3f0-2b3fba4fc804"
  }
]
```


SCHEMA


```
{
    "title": "GET List of Projects",
    "type": "array",
    "items": {
            "type": "object",
        "properties": {
          "identifier": {"type":"string"},
          "name": {"type": "string" },
          "description":{"type": "string"},
          "created_on": {"type":"string"},
          "analysis_id": {"type":"string"}
        }
    }
}
```

Figure A.2: API Appendix Page 2

GET Project Basic Information

`/api/project/<project_id>/`

DESCRIPTION

Get basic information about the project.

PARAMETERS

QUERY PARAMETER:

- <project_id>: The identifier of the project.

RESPONSE EXAMPLE

```
{
  "identifier": "e6b4577d-b80d-4819-a468-178167786140",
  "name": "El Ampli",
  "description": "",
  "created_on": "2017-11-19 13:41:29",
  "analysis_id": "9a33efa3-f199-4d22-b9a5-e5f857ca1c62"
}
```

SCHEMA

```
{
    "title": "GET Project Basic Information",
    "type": "object",
    "properties": {
      "identifier": {"type": "string"},
        "name": {"type": "string"},
        "description": {"type": "string"},
        "created_on": {"type": "string"},
        "analysis_id": {"type": "string"}
    }
}
```

Figure A.3: API Appendix Page 3

GET Project Details

/api/project/details/<project_id>

DESCRIPTION

Gets processed information about a restaurant. Processed information about contact restaurant data, and strong and improvement points about the project's restaurant.

PARAMETERS

QUERY PARAMETER:

- <project_id>: The identifier of the project.

RESPONSE EXAMPLE


```
{
  "identifier": "9a33efa3-f199-4d22-b9a5-e5f857ca1c62",
  "karma": 0.2092977862408325,
  "last_analysis": "2017-11-19 14:00:01",
  "restaurant_data": {
    "name": "El Ampli Barakaldo",
    "accepts_reservation": False,
    "star_rating": 5,
    "opening_hours": "...",
    "address": "Zaballa Kalea, 48901 Barakaldo, Bizkaia, Spain",
    "telephone": "665 64 32 03",
    "description": "Musica en vivo y cervezas artesanas."
  },
  "good_key_points": [
    {
      "identifier": "price",
      "name": "price",
      "karma": 0.3,
      "appearances": 1,
      "reviews": [
        ""
      ]
    }
  ],
  "improvement_key_points": []
}
```


SCHEMA


```
{
    "title": "GET Project Details",
    "type": "object",
    "properties": {
        "identifier": {"type": "string"},
        "karma": {"type": "number"},
        "last_analysis": {"type": "string"},
        "restaurant_data": {
            "type": "object",
            "properties": {
                "name": {"type": "string"},
                "accepts_reservation": {"type": "boolean"},
                "star_rating": {"type": "number"},
                "opening_hours": {"type": "string"},
                "address": {"type": "string"},
```

Figure A.4: API Appendix Page 4


```
                "telephone": {"type": "string"},
                "description": {"type": "string"}
            }
        },
        "good_key_points": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                  "identifier": {"type": "string"},
                  "name": {"type": "string"},
                  "karma": {"type": "number"},
                  "appearances": {"type": "number"},
                  "reviews": {
                    "type": "object",
                    "properties": {
                       "reference": "string",
                       "text": "string",
                       "karma": "number",
                     }
                  }
                }
            }
        },
        "improvement_key_points": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                  "identifier": {"type": "string"},
                  "name": {"type": "string"},
                  "karma": {"type": "number"},
                  "appearances": {"type": "number"},
                  "reviews": {
                    "type": "object",
                    "properties": {
                       "reference": "string",
                       "text": "string",
                       "karma": "number",
                     }
                  }

                }
            }
        }
    }
}
```

Figure A.5: API Appendix Page 5

GET Project Sources

`/api/project/sources/<project_id>/`

DESCRIPTION

Gets the name of the social networks set up in the project.

PARAMETERS

QUERY PARAMETER:

- <project_id>: The identifier of the project.

RESPONSE EXAMPLE

```
[
  {
    "social_network_identifier": "El Ampli Barakaldo",
    "name": "Twitter",
    "restaurant_id": "El Ampli Barakaldo",
    "country": "ES"
  },
  {
    "social_network_identifier": "230390187297922",
    "name": "Facebook",
    "restaurant_id": "https://www.facebook.com/El-Ampli-
230390187297922/",
    "country": "ES"
  },
  {
    "social_network_identifier": "ChIJayu_kXBaTg0R6sIoWp7K5cY",
    "name": "Google Places",
    "restaurant_id":
"https://maps.google.com/?cid=14332084170642735850",
    "country": "ES"
  }
]
```

SCHEMA

```
{
    "title": "GET Project Sources",
    "type": "array",
    "items": {
      "type": "object",
        "properties": {
        "social_network_identifier": {"type": "string"},
        "name": {"type": "string"},
        "restaurant_id": {"type": "string"},
        "country": {"type": "string"}
      }
    }
}
```

Figure A.6: API Appendix Page 6

PUT Request Update Social Media Data

/api/project/sources/<project_id>/

DESCRIPTION

Updates the information about a social network in a project.

PARAMETERS

QUERY PARAMETER:

- <project_id>: The identifier of the project.

REQUEST BODY:

- <social_network>: The social network name which identifier is going to be updated.
- <restaurant_id>: restaurant identifier from the social network.
- <geo_position>: the area from which data wants to be retrieved.
- <country>: country the restaurant is in.

INPUT DATA EXAMPLE

```
{
  "social_network": "(Twitter|Google Places|Facebook)",
  "restaurant_id": "El Ampli",
  "geo_position": "40.463667,-3.74922",
  "country": "ES"
}
```

SCHEMA (1) [If Google Places or Facebook]

```
{
    "title": "PUT Request Update Social Media Data",
    "type": "object",
    "properties": {
      "social_network": {"type": "string"},
      "restaurant_id": {"type": "string"},
      "country": {"type": "string"}
    },
    "required": ["social_network", "restaurant_id", "country"]
}
```

SCHEMA (2) [If Twitter]

```
{
    "title": "PUT Request Update Social Media Data",
    "type": "object",
    "properties": {
      "social_network": {"type": "string"},
      "restaurant_id": {"type": "string"},
      "geo_position": {"type": "string"}
    },
    "required": ["social_network", "restaurant_id", "geo_position"]
}
```

Figure A.7: API Appendix Page 7

PUT Request Analyze Project
/api/project/retrieve-data/{project_id}
DESCRIPTION
Downloads information about the restaurant from the set up social networks.
PARAMETERS
QUERY PARAMETER: • <project_id>: The identifier of the project.
RESPONSE EXAMPLE

PUT Request Download Restaurant Data
/api/project/analyse/{project_id}
DESCRIPTION
Request to the server the analysis of the project: (1) mix the contact information about a restaurant, and (2) find out the strong and improvement point of the restaurant.
PARAMETERS
QUERY PARAMETER: • <project_id>: The identifier of the project.

PUT Request Download Restaurant Data
/api/restaurant/nearby
DESCRIPTION
Retrieves a list of near restaurants.
PARAMETERS
QUERY PARAMETER: • latitude: User's latitude • longitude: User's longitude
SCHEMA

```
{
    "title": "PUT Request Download Restaurant Data",
    "type": "array",
    "items": {
    {
      "identifier ": {"type": "string"},
      "latitude": {"type": "double"},
      "longitude ": {"type": "double"}
    },
    "required": ["social_network", "restaurant_id", "geo_position"]
}
```

Bibliography

[1] Gerd Stumme; Andreas Hotho; Bettina Berendt. 'Semantic Web Mining State of the art and future directions'. In: *Journal of Semantics* (Berlin, Germany, 2005).

[2] *Semantic Web Layers*. https://www.w3.org/DesignIssues/Semantic.html. Accesed: 2017-09-27.

[3] *Linked Data*. https://www.w3.org/standards/semanticweb/data. Accesed: 2017-09-28.

[4] *RDFS VS OWL*. https://www.cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/rdfs-vs-owl/. Accesed: 2017-09-28.

[5] *Introduction to Data, Text, and Web Mining for Business Analytics Minitrack*. http://scholarspace.manoa.hawaii.edu/bitstream/10125/41284/1/paper0135.pdf. Accessed: 2018-01-08.

[6] *Bilbao, chosen as the 2018 best European city by the Academy of Urbanism*. https://aboutbasquecountry.eus/en/2017/11/08/bilbao-chosen-as-the-2018-best-european-city-by-the-academy-of-urbanism/. Accesed:2017-12-13.

[7] Elisa Claire Alemán Carreón, Hirofumi Nonaka and Toru Hiraoka. 'Analysis of Chinese Tourists in Japan by Text Mining of a Hotel Portal Site'. In: *ISIS. The 18th International Symposium on Advanced Intelligent Systems* (2017).

[8] *Flask*. URL: http://flask.pocoo.org.

[9] *TextBlob: Simplified Text Processing*. URL: https://textblob.readthedocs.io/en/dev/.

[10] *Machine Learning in Python*. URL: http://scikit-learn.org/stable/index.html.

[11] *Yarn*. URL: https://yarnpkg.com.

[12] *Webpack*. URL: https://github.com/webpack/webpack.

[13] *React Framework*. URL: https://github.com/facebook/react.

[14] *Performance of graph vs relational databases*. URL: https://dzone.com/articles/performance-graph-vs.

[15] *Neo4J Partners*. URL: https://neo4j.com/partners/.

[16] *React-Bootstrap*. URL: https://react-bootstrap.github.io/.

[17] *Motivation - Redux*. URL: https://redux.js.org/introduction/motivation.

[18] *Travis CI - Test and Deploy Your Code with Confidence*. URL: https://travis-ci.com.

[19] *OpenER*. URL: http://www.opener-project.eu/.

[20] J. Villena-Román, S. Lana-Serrano, E. Martínez-Cámara and J.C. González-Cristobal. 'LIBSVM Workshop on Sentiment Analysis at SEPLN'. In: *Procesamiento del Lenguaje Natural, 50* (2013). http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/4657.

[21] J. Villena-Román, J. García-Morera, S. Lana-Serrano and J.C. González-Cristóbal. 'TASS 2013-A Second Step in Reputation Analysis in Spanish'. In: *Procesamiento del Lenguaje Natural, 52* (2014). http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/4901.

[22] J. Villena-Román, E. Martínez-Cámara, J. García-Morera and S. Jiménez-Zafra. 'TASS 2014-The Challenge of Aspect-based Sentiment Analysis'. In: *Procesamiento del Lenguaje Natural, 54* (2015). http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/5095.

[23] E. Martínez-Cámara, M.A. García-Cumbreras, J. Villena-Román and J. García-Morera. 'TASS 2015-The The Evolution of the Spanish Opinion Mining Systems'. In: *Procesamiento del Lenguaje Natural, 56* (2016). http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/issue/view/218.

[24] *WeLive*. URL: https://dev.welive.eu/.