Università
Ca'Foscari
Venezia

Master's Degree programme – Second Cycle (*D.M. 270/2004*)
in Informatica – Computer Science

Final Thesis

–

Ca' Foscari
Dorsoduro 3246
30123 Venezia

Destination Monitor design for benchmark evaluation of Tourism Product

Analysis of indicators and impacts

Supervisor
Ch. Prof. Agostino Cortesi

Graduand
Giulia Schiavon
Matriculation Number 826788

Academic Year
2015 / 2016

ACKNOWLEDGEMENT

First of all, I wish to acknowledge my supervisor Agostino Cortesi, without him starting and developing this Master's Degree Thesis would not have been possible.

I would also like to thank my sister Claudia and my brother-in-law Michele, who welcomed me into their home during the development of this thesis and corrected my mistakes and my terrible English.

I would also thank my boyfriend Alessio, who trusted in me during these two years and helped me in every moment.

I must express my very profound gratitude to my parents, who supported me during my years of study.

Abstract

In this Master's Degree Thesis we will conjecture an algorithm to define the *Appeal* in a Destination Monitor context.

The goal consists in identifying a weighted point of view from the customers' feedback and statistical information. The final result is to be used as a comparison between *Tourism Products* of the same kind. The algorithm is based on defined indicators and the weighted average, a special counting in which each component is weighted differently.

The last contribution is the design of a database used to save the data retrieved from the Web and from *Tourism Product* sources, as well.

Table of Content

Index of Figures

Index of Tables

Index of Code

Introduction

Destination Monitor plays an important role in the field of the tourism. This paper will present an idea of development of a system, devoted, firstly, to tourism management, and secondarily to visitors' and clients' satisfaction. When we talk about tourism management, we are talking about pubs, restaurants, and hotels owner, or everything else that contributes to the improvement of tourism.

Day by day, we can see that tourism is a pillar on which a nation is based on.

The development of the theory we are going to suggest is based on an apprenticeship at CISET[1], tourism department of Ca' Foscari in Venice, Italy. In this period, we understood the significant idea of the creation of a system that can retrieve each information about the *Product Manager*'s activity, in both negative and positive ways, and compare the relative *Tourism Products* to similar others. It is important to underline that

[1] CISET is International Centre of Studies on Tourism Economics

nowadays, there are a lot of systems that take the info and give an account of the total situation in the World Wide Web in similar ways.

The goal of this Master thesis is the definition of a final *Appeal* of a *Tourism Product*: nowadays, we can see that there are some services and systems that create lists with the best and the worst existent products; without differentiating them by typology and without considering some external conditions and variables, as well. So, *Appeal*, is our final result, and we can define a point of view, crossing ten different parameters.

The final *Appeal* is a value that is the indicator towards to improvement of the product in a determinate component, or it is a comparison between other products with similar in features. In this way, we can define a possible benchmark determining the progress of the *Tourism Product*.

The idea of this project is explained in three steps, in which we will show different sides of the final system.

- The first one is the data retrieval from the Web, by means of opportune indicators and appropriate algorithms[2];

[2] reference [13]

- The second step consists in the client analysis: what the customers want and how they want it, both online and offline[3];

- The third one is the union of the previous outputs, defining new indicators and a code of the final algorithm, determining the *Appeal*.

The paper is divided into five chapters; each of them precisely describes the steps of the whole project.

1. Chapter one defines the Destination Monitor – what it is and how it is used in real life; it explains the actors co-existing in this dimension and their actions.

   Eight real examples of the use of existent systems and their implementation will be presented: it is the starting point that allows a comparison to new our proposal[4].

[3] references [23], [24], [25], [26]

[4] references [5], [6], [7], [8], [9], [27], [28], [29], [30], [31], [32]

2. Chapter two is the first step of the final *Appeal*.

   We will discover "when" and "how" the data are retrieved. We will analyze two famous algorithms, which are specific to this context – some execution examples are shown later in this chapter.

   In addition, we will study the indicators request by *Tourism Products*.

3. In Chapter three we will basically describe the second phase of the *Appeal* definition.

   We will see the Destination Monitor from the client side: in fact, without clients (namely tourists), we do not have any feedback and new potential customers of *Tourism Product*. So, we will analyze what features tourists look for and how they evaluate them.

4. After the previous analyses, in Chapter four we will have the possible solution for the *Appeal* definition. It is the heart of the whole hypothesis; in fact, we will define the indicators that are used in a weighted average, whose result is the *Appeal*.

The indicators are the results of the study carried out during the training in CISET and what *Product Managers* want. We will have ten possible metrics of comparison.

5. In the final Chapter five we can theorize some possible optimizations, implementations and future works.

In the paper there are references to existent services and interviews to *Product Managers*, who helped us in understanding what they wish a system would analyze the final *Appeal*. The result is taken as a reference of the benchmark in the range of the improvement.

1. Destination Monitor

Before entering in the detail of our new system, it is important to define some basic concepts, such as what tourism is and what the requests by both customers and owners are.

When thinking about tourism, what immediately comes to our minds is the activity of touring, especially for pleasure, but there are many other elements that define tourism.

It is important to know and analyze tourism because it is connected to several fields – like economy, society and environment, which all impact on the territory and its develpoment.

Another reason for studying tourism is provided by personal interconnections: we can see how the relationships are modified in a given territory – locally, nationally and internationally, as well the economy and other components.

Tourism is characterized by constant evolution, in fact destination trends rapidly change synchronically and asynchronally.

This thesis focuses on a branch of tourism, namely Destination Monitor[5]: we can see that the reputation and how a *Tourism Product* appears with respect to concurrent others are central.

With this respect, we have to define some concepts, like touristic actors in a Destination Monitor. A Destination Monitor is defined as a system that studies the progress of a destination, using indicators.

1.1 Actors in Tourism

It is common place that the actors which appear in a touristic environment are tourists and destinations, only.

A tourist is a person who (usually) travels for pleasure, sightseeing and thus staying in hotels; a destination is defined as the predetermined object of a journey.

But, these definitions are given by the fact that, for the majority of people, tourism only means a holiday resource; on

[5] references [1], [2], [3], [4]

the contrary, the destination is very often related to one's work place, to one's health, to family matter, or just to a step of a whole holiday. So, we can define tourism as a combination of *Tourism Producers*, *Distributors* and *Consumers*.

We will see these elements in detail: the following *Figure 1* shows the aformenentioned actors' features and their distinctions.

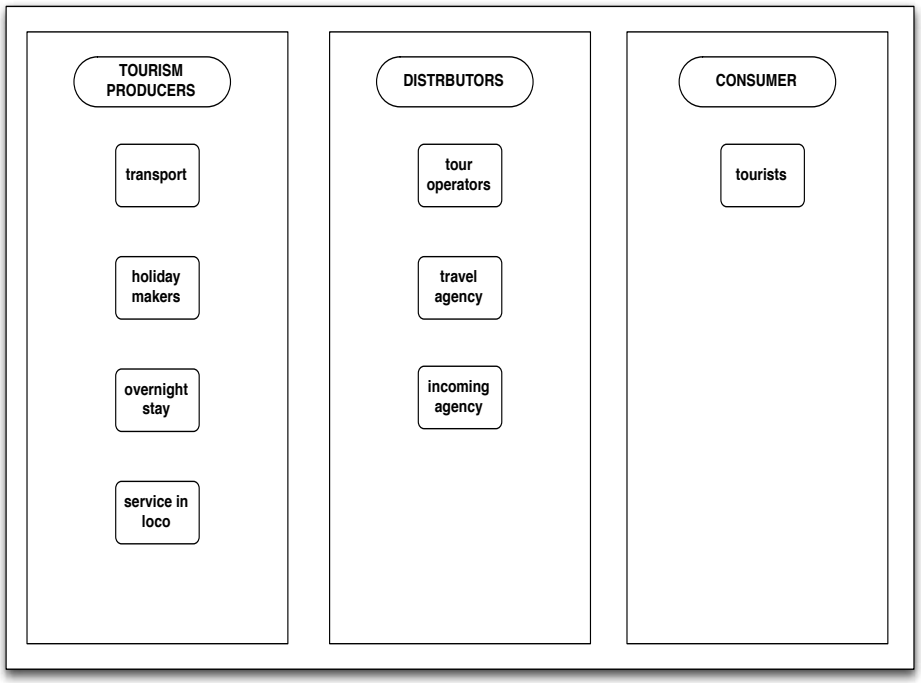

Figure 1 ~ Tourism

We are paying a greater attention to the first actor: *Tourism Producers*. But it is important to get acquainted to the others, as well.

Distributors and *Consumer* are not part of the Destination Monitor and *Appeal* does not focus on them; but, in order to get a better view of the target in our case study, we should make an overview of all these components. So, the following subchapters show the categories in detail.

1.1.1 *Tourism Producers*

The first actors present in tourism are *Tourism Producer*s: a tourism producer is the destination. Destination is not only the "city" that we visit, but it can also involve other kinds of *Tourism Producers*, which are divided into four typologies.

1. The first one is *Transport*.

   It is interesting to note that the *Transport* is divided into two different categories: the one which allows us to get to

the destination, and the one which is present in the destination.

So, *Transport* can include public transport (plane, train, bus, boat, etc.), or private means of transport, like cars. We can see that this information is important when we define the indicators about the *Appeal* definition.

2. The second typology is named *Holiday Makers*, which consists of pre-packaged travels. Two examples, that are cruise ship and Theme Parks, show what *Holiday Maker* means.

   When tourists get involved in this destination typology, they have fun through the attractive which is present in it. Whether they chose a Theme Park, they know that they can get restaurants, shops and rides within the destination. If they chose a cruise ship, they know that their stay is decided and pre-packaged by others.

3. The third kind is featured by hotels or camping sites, that are the *Overnight Stay*.

When *Overnight Stay* are meant as hotels, they provide tourists with overnight facilities and also kitchen equipments.

4. The fourth and last typology is the *Services In Loco*.

   The *Services In Loco* are all the touristic services present in a determinate destination. They include restaurants, shops et similia. *Service In Loco* is sometimes particular and characteristic of the place, while others are chains present everywhere.

It is important to stress that these four typologies are not perfectly subdivided, overlaps are possible. In this thesis we try to divide them as clearly as possible, because typologies help us create the final *Appeal* of the *Tourism Product*, and we can compare *Tourism Products* basing on the same parameters.

We can see that there are mash ups among the products. An example is provided by *Holiday Makers*: there are some *Services In Loco* which are built especially for the structure; so, we have two different *Tourism Products* to compare in the same site.

In Chapter four we will see how typologies are used.

1.1.2 *Distributors*

A distributor is an entity that provides tourists with the tools to arrange their travel. Tourists can plan a journey in several ways – relying on different *Distributors* (agencies in the Web…).

There are three sub-categories.

1. *Tour Operators*.

   They combine components to create a package holiday. They advertise and print brochures to promote their products, holidays and itineraries. The journey, the location and many other elements are decided by a third party.

2. *Travel Agencies*.

   They are slightly different from the first distributors. In fact, we can define a *Travel Agency* as a retailer that provides with travel and tourism related services to the

public on behalf of suppliers. In this case, the holiday is defined in every small details, starting from the transport, to get to the daily program.

3. *Incoming Agencies*.

   Different from the other distributors, in the fact they are located on the incoming destination: they prepare packages in which they define programs and what to visit in the incoming area.

All of these kinds of distributors can be in offline and online.

1.1.3 *Consumers*

Eventually, we make a brief overview of the last actor present in the tourism. A *Consumer* is defined as a person that uses *Tourism Producers* and *Distributors*.

Typically, the *Consumers* are the tourists; but we have to differentiate them into two categories:

1. A *tourist* is defined as someone leaving from a place to another, for at least twenty-four hours time but not over one year;

2. A *hiker* is someone visiting a target destination for no more than twenty-four hours, and without staying overnights.

It is important to underline that *Tourist* is not a worker: he/she spends his/her time for pleasure, during holiday time.

1.2 Pre-existent Destination Monitors

Several data retrieval tools are available, but our case study focused on the *Appeal* evaluation, given by eight services in eight Italian Regions: in detail, we study how they work and what they want.

The outputs were provided by regional agencies via e-mail exchange.

Those agencies found it very important to create a Destination Monitor service, as it will impact with economy. We make a brief overview of the system used by the aforementioned Italian Regions.

To simplify the concept, we can divide the Regions into two different groups, because some of them has similar attitude.

The first group is composed by Tuscany, Piedmont[6], Liguria, Emilia Romagna[7] and Lombardy[8]. In this group the Regions are not aware of how the data for defining Destination Monitor are

[6] reference [30]
[7] references [9], [31]
[8] reference [32]

obtained: they are supported by real services that analyze Open Data and then the outputs are saved in the Region's database. The first group did not create or program any services, but used existent services that retrieve information and data from public database, Statistical National Institute and the Web (including social networks). Unfortunately, we do not have algorithms or info on how the data were retrieved. The Regions all retrieve data from Open Data.

In this work we focus on the second group, composed by Puglia[9], Trentino[10] and Veneto – with particular attention to Venice[11].

Differently from the first group, those Regions all created a service that retrieves data from Open Data and specific indicators to adopt in the analysis, as well. They created a service basing on what the features of tourism are – e.g. tourist flows, customers inquieries, gemorphological diversity…

Now we will study these features, to understand their work and compare them.

[9] references [5], [6], [7], [8], [29]
[10] reference [28]
[11] reference [12], [27]

The first system that we explored was created by Puglia Region; it is divided into two different phases.

1. The first step retrieves data by a telematic system: basically, it allows the receiving structure to directly send data to the central server. Every two weeks there is a data storage, so that they are always updated.

2. The second step consists in sending data to ISTAT[12] to check the statistics. Once ISTAT approves what is sent, the data are saved in the Region's own database, creating and updating Puglia Open Data.

In Puglia, the *Product Managers* of various *Tourism Products* send information and data to the Region server and this information and data are sent to ISTAT to be checked in a second moment.

Now, we analyze the second system. Trentino Marketing is a branch of Trentino Region and works following two phases.

[12] Istituto Nazionale di Statistica Italia. It is the Italian structure in which each commercial structure sends its statistics.

We do not focus on how the service checks the clients' (namely tourists) feedback. Their concern in mainly the *Overnight Stay*.

The retrieval is carried out by an external service named *Develon*: this project analyzes data from the hotel itself because *Product Managers* themselves insert their statistics. It is important to underline that data are not publicly revealed, as they are considered sensitive information, for this reason they are used only in an anonymous way.

As far as our work is concerned, the interesting part is the feedback retrieval from social networks: Trentino Marketing employs a program (named *TrustYou*) written by a Master's Degree student who proposed his program in his thesis. In this service, there is a crawler which retrieves the most frequent keywords related to tourism world from several social networks and inserts them in a list.

The last service of this group, is named *Venice Project Center*: this dashboard was created specifically for the city of Venice. The project is written by the students attending the Worcester Polytechnic Institute[13] who also supplied with online

[13] Private research university in Worcester focusing on the instruction and research of technical arts and applied sciences.

tutorial. *Venice Dashboard* is a web application developed in order to display information about Venice in real time, using individual modules or widgets. Each widget collects publically-available information from existing web sites, using mash up techniques or API's.

After the data retrieval, the information is saved in the Venice Open Data, which becomes public and usable by other entities.

1.3 Purpose of this thesis

Basing on the aforementioned analyses, we came up with the following conclusion.

Examining the development of the *Appeal* appears to be crucial, but services mash up the result and the outputs, without understanding whether a component is more important or relevant than others. Thus it is unlikely to get omogenous data.

This situation leads to the misleading comparison of structures which have dissimilar features.

In this perspective, in the following chapters we will try to determine the *Appeal*, using indicators and outputs based on algorithms, which are specifically designed for our aim.

The result will have an important impact in the definition of the *Tourism Product* and we will be able to delineate a benchmark to be used by *Product Managers* to compare similar activities.

In the fourth chapter we will detail how the comparison is made.

2. Service-manager system

In this chapter we will show the system we employed to retrieve the data in the Web and its relative algorithms.

This system works in backstage, analyzing data in the World Wide Web. The main idea is based on the retrieval, not only of the Open Data[14], but of sensitive data and then they are crossed in a weighted mean.

The chapter is divided into two parts: the first subchapter shows how this system works, illustrating the codes employed; the second part presents the indicators to be retrieved.

Retrieving the Open Data is quite easy as there are not copyrighted issues or anyone's control, so they are free of use.

The final goal of this thesis consists in looking for an *Appeal* of the *Tourism Product*.

[14] They are freely available to everyone to use and republish as they wish.

It is important to remember that the case study is concerned with the Destination Monitor, and we can find several interesting roles and consequences of it. In fact, we can have different results based on the choice of the Destination Monitor.

2.1 Data retrieval

This paragraph shows the data retrieval and the relative algorithms adopted. *Figure 2* is a simplified illustration of how we can retrieve the data in the Web.

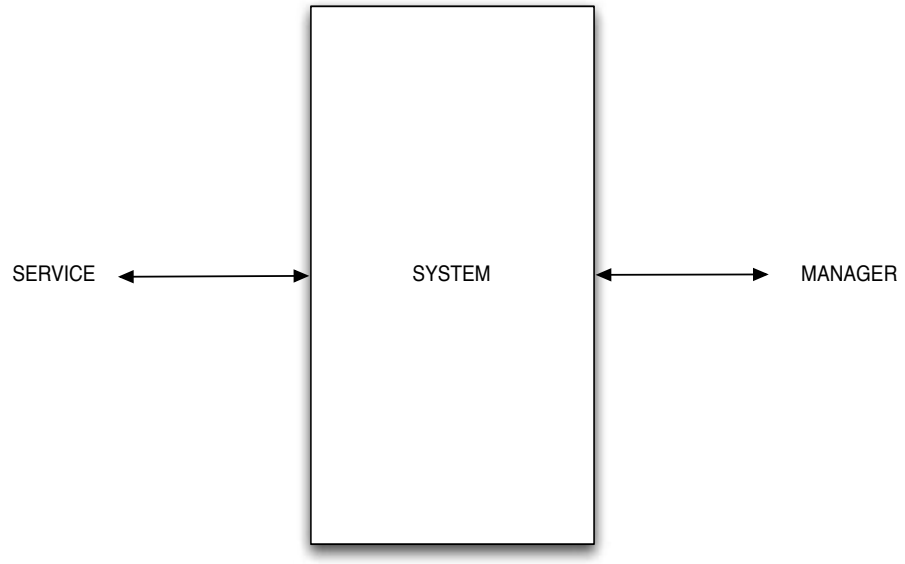

Figure 2 ~ Data retrieval

In this schema we can see that *Service* and *Product Manager* talk together: the double arrows show that *Service* writes and reads what the *Product Manager* does, and vice versa.

Service retrieves the data from social networks or tourism portals, in which customers can leave a feedback or comments: in this way, everyone can say his/her opinion about the *Tourism Product*.

Figure 3 shows the *Product Manager* actions.

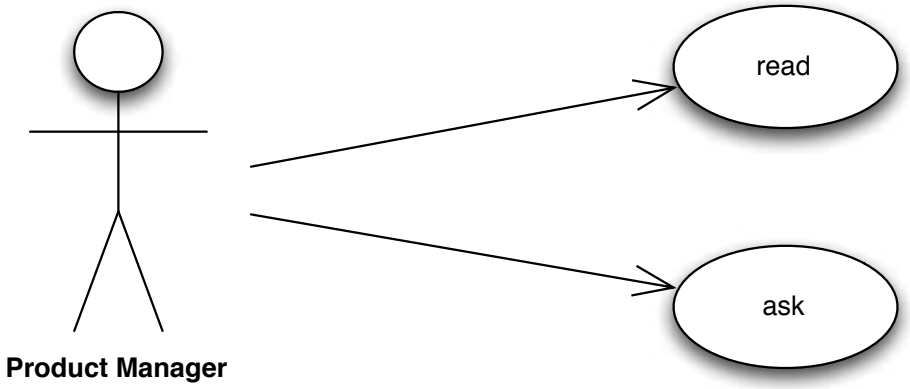

Figure 3 ~ UML schema Product Manager

We analyze the different actions that *Product Manager* can do:

- **READ**

  Product Manager reads the service results;

- **ASK**

  Product Manager asks some information about his/her *Tourism Product*.

System is the hearth of the whole operation; in fact, it processes the data given by the algorithms and shows the final progress of the activity.

In the following paragraphs, we will see the *Crawler* algorithm and the *PageRank* algorithm[15], with some examples.

We created the last algorithm, *Worldrelations*, which is a particular algorithm that creates links among the keywords.

[15] reference [10], [13]

2.1.1 *Appeal*

The *Appeal* is defined as the power to attract interests, and it determines the choice of a destination. *Appeal* is the goal of every service or system addressed to *Product Manager*.

We can define the *Appeal* into two different categories[16]:

1. The first one consists in the *Explicit Appeal*, published on different web portals, with the goal of understanding how positive and negative feedback can influence the destination choice;

2. The second one is named *Implicit Appeal*, that allows *Product Manager* to evaluate *Tourism Product*, using data retrieval from statistic study.

Combing the previous parameters, and using a weighted average, we can define the final *Appeal*.

[16] reference [1]

2.1.2 *Crawler*

We see the first algorithm used in the data retrieval. A *Crawler* is a computer program that is capable of performing recursive searches on the World Wide Web.

The *Crawler* systematically crawls pages and looks at the keywords and links within the page, then returns that information to the search engine's server for indexing.

The code in Python[17] language will be shown in Appendix A.

In the following *Table 1*, *Table 2*, *Table 3* and *Table 4*, we present four executions and relative outputs of the algorithm.

The execution is divided in two steps: the first step is the choice of a web site, the second step is search of the occurences of some given keywords, in the web site.

The results are shown in the following tables; each of them has three attributes.

[17] Python is chosen because is similar to pseudo code. It is particular because it is a dynamic programming language.

1. *Web Site*

   It is the first one, it is the web site from which we are able to retrieve the occurences of the keywords;

2. *Lev*el

   The *Crawler* can search keywords in different levels of the web graph. In these specific examples we used one or two levels;

3. *Keywords*

   It contains all the keywords with the relative occurences.

Web Site **http://www.legambiente.it/temi/turismo**

Level	1	
Keywords	Roma	444
	storico	51

Table 1 ~ Crawler example 1

Web Site **https://www.tripadvisor.it/Hotel_Review-g187849-d229090-Reviews-Hotel_Berna-Milan_Lombardy.html**

Level	1	
Keywords	qualita	33
	servizio	287

Table 2 ~ Crawler example 2

Web Site **https://www.tripadvisor.it**

Level	2	
Keywords	hotel	128345
	Mestre	463

Table 3 ~ Crawler example 3

Web Site **http://www.volareweekend.com/it/offerte-voli/capodanno/capodanno-low-cost.html**

Level	1	
Keywords	volo	407
	notte	210
	citta	17
	divertente	100
	cultura	347
	benessere	198

Table 4 ~ Crawler example 4

2.1.3 *PageRank*

In this paragraph we study the *PageRank* algorithm: it is used to rank web sites in their search engine results. *PageRank* works by counting the number and the quality of links to a page to determine a rough estimation of how important the web site is.

The underlying assumption is that more important web sites are likely to receive more links from other web sites. Actually, this algorithm was created by Google[18].

Its work is easy: it exploits incoming links from popular pages to raise the rank of the pages themselves. We get the position of the page, and we know how much that topic is quoted. This algorithm helps understand what favorite attributes are when clients are looking for a destination.

The *PageRank* algorithm sees the Web as a directed graph with the pages being nodes and hyperlinks being connections between those nodes. It can be used to rank the nodes of any kind of graphs (including undirected ones) by importance.

This description uses graph terminology and only shows how it is done for a directed graph such as the web graph.

[18] The PageRank citation ranking: bringing order to the Web, 1998

While it is accurate to say that *PageRank* will tell us the *importance* of each page, a more accurate definition is that *PageRank* assigns a *probability* to each page. Specifically, the *PageRank* value of a page is the probability, between 0 and 1, that someone, surfing the page by clicking links randomly, will end up on that page.

With *PageRank* we have a measure of the rank prestige: it forms the basis of most web page link analysis algorithms.

In Appendix B we will present the *PageRank* code in C[19] language.

The algorithm has a file in txt format as input in which there are two columns: each of the element of the first column is the starting point and the second column represents the arriving node.

The output is the a vector which presents the prestige of each link.

[19] C is a structured and procedural programming language that has been widely used for both operating systems and applications.

Final P Vector:
0.022 0.017 0.008 0.006 0.013 0.004 0.007 0.018 0.004 0.005
0.008 0.020 0.004 0.006 0.007 0.014 0.004 0.013 0.009 0.006
0.006 0.009 0.007 0.015 0.004 0.004 0.005 0.006 0.028 0.004
0.017 0.004 0.035 0.034 0.006 0.011 0.006 0.016 0.023 0.007
0.004 0.014 0.020 0.010 0.018 0.006 0.005 0.016 0.004 0.011
0.019 0.009 0.004 0.004 0.027 0.011 0.008 0.008 0.012 0.005
0.012 0.005 0.019 0.004 0.007 0.019 0.019 0.004 0.004 0.004
0.005 0.017 0.012 0.009 0.004 0.004 0.007 0.005 0.008 0.014
0.006 0.004 0.004 0.005 0.011 0.030 0.011 0.004 0.004 0.012
0.008 0.004 0.006 0.006 0.004 0.023 0.004 0.008 0.005 0.007

Table 5 ~ PageRank example

The previous *Table 5* shows an example of *PageRank* with a graph composed by 100 nodes as input.

2.1.4 Wordrelations

In this paragraph we present an algorithm that is a combination of the previuos algorithms. From a given keyword provided by Google API[20], we search if there is a link to other keywords from the same set.

In Appendix C we will present the code written in Python language.

The results are shown in the following tables, through two parameters:

1. The first one is *Keyword*, that is the initial keyword we start searching;

2. The second one is *Wordrelations* and it is the results of the algorithm. We see the occurences of the words starting from a keyword.

[20] Google APIs are sets of application programming interfaces developed by Google which allow communication with Google Services and their integration to other services.

Keyword	**Capodanno**	
Wordrelations	Montagna	17
	Spiaggia	4

Table 6 ~ Wordrelations example 1

Keyword	**Ferie**	
Wordrelations	Montagna	0
	Campeggio	0
	Mare	7

Table 7 ~ Wordrelations example 2

Keyword	**Hotel**	
Wordrelations	Colazione	1
	Spa	3397
	Parcheggio	1

Table 8 ~ Wordrelations example 3

Keyword	**Vacanza**	
Wordrelations	Roma	7
	Venezia	1
	Trieste	0

Table 9 ~ Wordrelations example 4

2.2 Indicators definition

According to CISET standard[21], we define now five indicators, that are used to determine the *Appeal* of a *Tourism Product*.

We analyze the definition of these indicators. They are presented through *Table 10*, *Table 11*, *Table 12*, *Table 13* and *Table 14* in which there are three cells.

I. NAME

The first cell shows the name of the indicator. The name is the identification ID for each indicator.

II. DEFINITION

The second attribute is the definition of the indicator. The definition helps us understand what it is and what its goal during the data retrieval is.

[21] references [1], [2], [3], [4]

III. IMPACT

The third cell is particular because it explains what the impact in the *Appeal* search is: it takes the information from the client's sides and the *Product Manager's* side.

We can see how the *Appeal* can change in front of determinate characteristics.

It is important to remember that these indicators are adopted to evaluate the Destination Monitor.

NAME	**Brand**
DEFINITION	Particular product or a characteristic that serves to identify a particular product. Using this term we can immediately think of the *Tourism Product*, and so we can distinguish it through the concurrency.
IMPACT	This indicator is important to guarantee a certain confidentiality with customers. The client knows the product and, to some extent, acts as he/she knows every part of the product. The role of this indicator consists in improving the brand, so every client knows it, as a friendly brand. If the client knows the brand, it is more likely that the client sponsors it to friends.

Table 10 ~ Indicator – BRAND

NAME	**Products**
DEFINITION	A set of tangible or intangible attributes of a service; it is usually obtained by a production process or a creation from initial resources with the goal of improving the final value.
IMPACT	The impact of this indicator gives an account of the *Tourism Product* activity trend. From the *Product Manager*'s side it is important to understand the impact because it shows what clients ask, and what to change or to improve. From the client's sides we can see that tourists choose a product with respect to another.

Table 11 ~ Indicator – PRODUCTS

NAME	**Performance compared to the same type destinations**
DEFINITION	Total progress of the activity, including the economic and, sometimes, political aspects. The results give an overview of the progress, with respect to other structures present in the same destination. The indicator consists in a comparison between similar structures in the same destination.
IMPACT	The main impact of this indicator is the concurrency control: in fact, a *Product Manager* can control the progress of the competitors at any time, so he/she can improve himself/herself. The clients have the possibility to choose the *Tourism Product* comparing it to others, in the same destination.

Table 12 ~ Indicator - PERFOMANCE COMPARED TO THE SAME TYPE DESTINATION

NAME	**Progress of investments and dynamic of the offer, expenses and use of attractors**
DEFINITION	Progress of the activity using the clients' feedback. What clients want and how they want it.
IMPACT	The indicator is related to the *Product Manager*, since it shows the progress of the *Tourism Product* using the feedback and the comments of customers. Using this indicator the *Product Manager* can immediately understand what clients do, and, can modify the features of the *Tourism Product* to attract clients.

Table 13 ~ Indicator - PROGRESS OF INVESTIMENTS AND DYNAMIC OF THE OFFER, EXPENSES AND USE OF ATTRACTORS

NAME	**Accessibility**
DEFINITION	Possibility to be accessible to other people.
IMPACT	This indicator has two different impacts. 1. The first performance is on the accessibility of the web site, and it is addressed to the *Product Manager*: it makes the *Product Manager* himself/herself understand if the web site is accessible and user-friendly. 2. The second performance is on the phisycal accessibility of the *Tourism Product*. It is addressed to the clients, since it involves some parameters that identify whether the destination is easy to reach – e.g. considering public transport.

Table 14 ~ Indicator – ACCESIBILITY

3. Manager-client system

In this chapter we will present the system which links product *Product Managers* and *Customers*. In this system we see that *Product Managers* and clients can speak together, through message exchanges.

In a social network everybody can leave a comment or an opinion about his/her life, or in this specific case, about a *Tourism Product*.

Basically, the idea consists in the creation of a social network in which the *Product Manager* can describe in detail his/her activity, and the client can read and, if he/she wants, leaves a comment.

The chapter is divided into three sections: the first one is the behaviour of the system, analyzing the clients' and *Product Managers'* actions; the second one is the clients' requests. The last section is concerned about the keywords which are used in

the final part of the project: in fact, knowing the clients' requests, we can get the weighted average and define the final *Appeal*.

3.1 *Product Manager-Client* relationship

The technical job of the *Product Manager-Client* system involves the actors who exchange messages[22].

Figure 4 simplifies the behaviour of the *Product Manager-Client* system.

It is interesting to note that in this system each *Product Manager* can describe his/her activity product, so the *Product Manager* is also the *Product Manager* of his/her web site.

[22] references [15], [16], [17], [18], [19], [20], [21], [22]

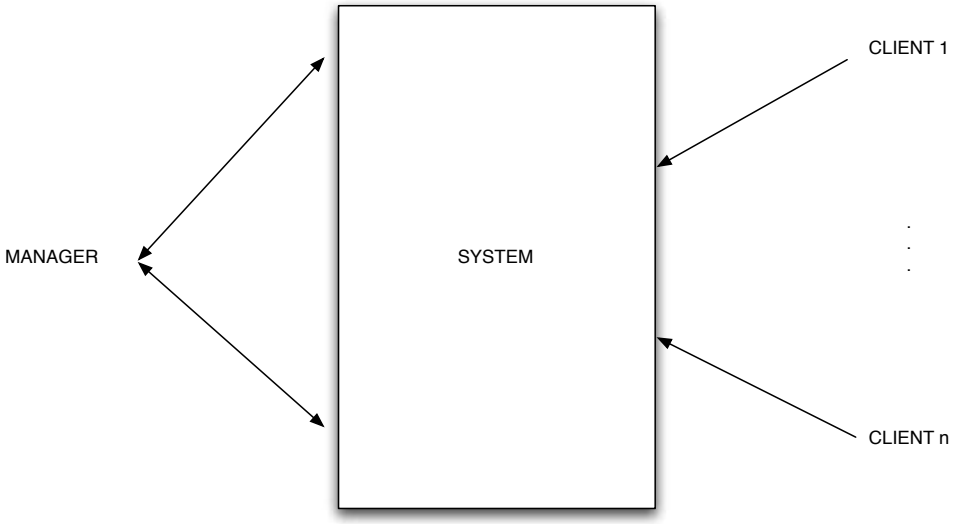

Figure 4 ~ Product Manager-Client relationship

Between the *Product Manager* and the *System* there are double arrows because the *Product Manager* asks, answers and writes the page, while *Clients* can read and write (annotate, estimate, book) on the page, but the *Client* can not perform the ask action.

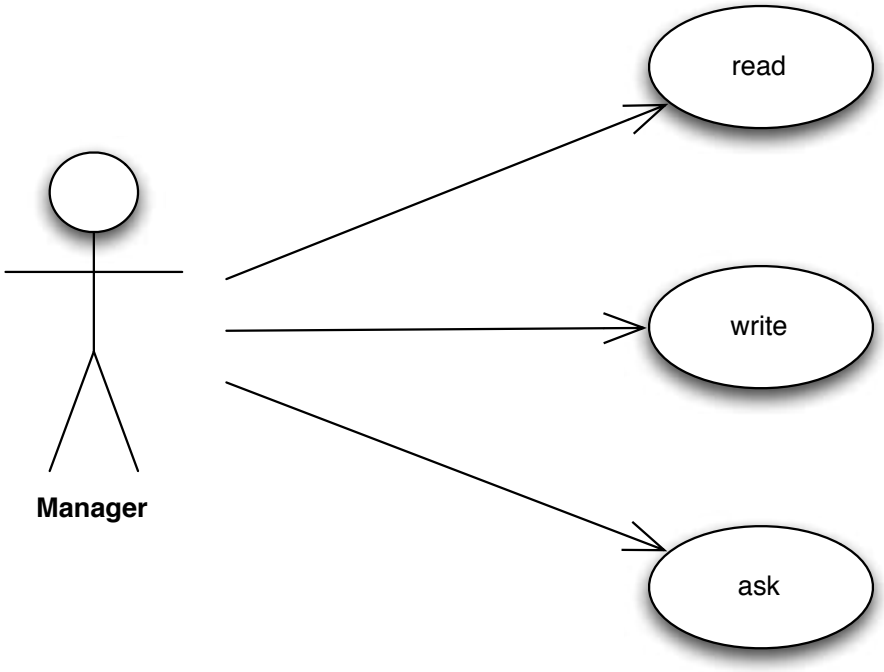

Figure 5 ~ UML Product Manager

Product Manager has three possible actions:

- **READ**

  The first action allows the *Product Manager* to read clients' feedback, so he/she can have the idea of what clients require from his/her *Tourism Product*;

- **WRITE**

  Product Manager describes his/her activity, and upgrades the web site with news and advisement, he/she answers clients' requests, as well he/she maintains a strict relationship with the client;

- **ASK**

  In this action, *Product Manager* interrogates the system, e. g. asking what is the percentage of a certain keyword.

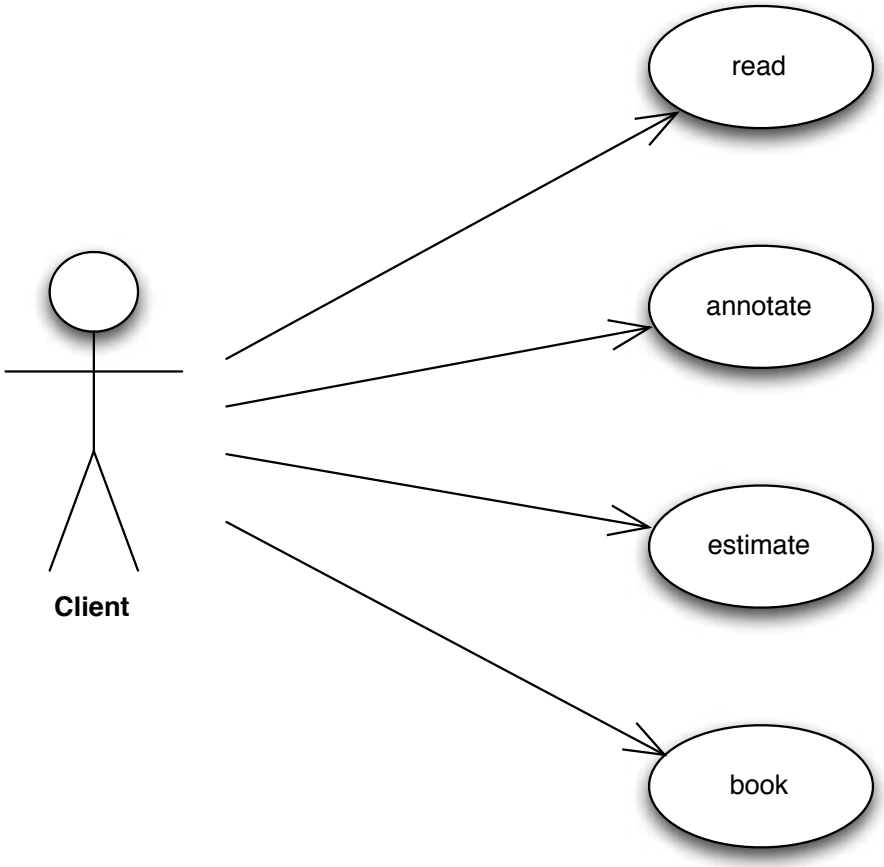

Figure 6 ~ UML Client

The client has four actions:

- **READ**

  The *Client* reads the description and the *Product Manager*'s answers;

- **ANNOTATE**

  The *Client* writes his/her positive or negative opinions in the page, and asks the *Product Manager* some information;

- **ESTIMATE**

  This action is related to the evaluation of the activity, basing on several criteria. The activity is assigned one to five points;

- **BOOK**

  The last action allows the client to book a *Tourism Product*.

3.2 User-friendly requests

During the training at CISET, we collected information from different *Product Managers* about what their expectations were, with respect to the system we are proposing.

As far the *Clients'* requests are concerned, they are the results of the *Wordrelations* algorithm.

3.2.1 View from *Product Manager's* side

We interviewed four *Product Managers*: they work in two different environments (*Overnight Stay* and *Service In Loco*), but each of them showed common requests for a monitoring system.

The first interview was to a Restaurant Manager: in this context we can see that the restaurant is in a small town, quite far from the city center; so the first problem is where it is located. This *Tourism Product* has, however, some clients,

because its main strength is the strict relationship between the *Product Manager* and *Customers*: friendship allows a big word-of-mouth with other possible clients.

The word-of-mouth is the biggest strength also because this *Tourism Product* has its interest in tradition: within the last thirty years, the restaurant has a little upgraded, maintaining the focus on the food quality and the territory.

The second interview was to a bathhouse: the beach is a strong attraction during the summer, so there are many customers every year, especially during the weekends. It is interesting that the *Product Manager* keeps a relationship with the loyal customers even during winter time via emails, letters and social networks.

The third interview was to a young adult who manages a summerhouse near the beach, a few kilometers far from the city center (Venice). In this interview, the territory was mentioned several times: he thinks that tourists are attracted by the touristic city which is a few kilometers far from the *Tourism Product*. As well as the second *Product Manager* which was interviewed, this *Product Manager* emphasizes the strict

friendship with clients, and he also wants to maintain the relation during the winter time, when his activity is closed.

In order to allow a bigger and new clientele, the *Product Manager* wishes that news and improvements would be advertised.

The last interview was to a man who worked for a renowned hotel chain in Italy. He has been working for a long time, he has discovered that Internet is a new source of clients as inside web sites it is possible to show the hotel's improvements and news.

However, the word-of-mouth is still a strong way of advertising in a *Tourism Product*: if a customer is happy, then the reputation is good, hence other clients are attracted, otherwise fewer and fewer customers are likely to book. It is interesting to note that in this context, the friendship between the client and *Tourism Product* is a key point. The focus for this *Product Manager* is on the touristic city center.

3.2.1.1 Keywords

From the previous four interviews, it is interesting to note there are seven keywords that summarize what *Product Managers* think is the best way to maintain and increase the *Appeal* of their *Tourism Product.*

1. **FRIENDSHIP**

   It is a little odd that friendship is the first and most popular keyword, as it is present in all the interviews.

   It is important to maintain a close bound with the clientele, so that customers will be likely to return in future.

2. **INNOVATION**

   Although, innovation is not always a good idea, as changing the brand could increase the risks of losing customers, three of the interviewed *Product Managers* think that it is one of the strenghts of a *Tourism Product*: innovation shows the continuously wish of modernity.

3. **SOCIAL NETWORK**

   Today, a vaste part of the population owns a social network account, *Tourism Products* must be online, so that they can be almost instantly connected to their clients.

   The *Product Manager* can illustrate his/her activity, describing it and answering any questions via social network.

4. **WORD-OF-MOUTH**

   It is the oldest form of advertisment, born in the ancient times. If someone is satisfied, his/her friends or acquaintences are likely to be informed; and vice versa, they are likely to be advised not to book the structure.

   Nowadays, it is done immediately through a *tweet* or a *like*.

5. **DESTINATION**

   The importance of the destination was stressed by all the *Tourism Products*. Actually, tourists book their holidays basing on the popularity of the territory, choosing hotels,

restaurants, Theme Parks and so the choice is directly related to the destination.

6. **WEB SITE UPGRADES**

   Customers choose a *Tourism Product* also relying on the quality of the web site, in terms of usability and accessibility.

   The code must be upgraded to new technology, as nowaday the majority of customers own a smartphone and tend to surf the Internet using it, so every device must read the information in order to find the closing days, opening hours, prices, facilities…

7. **TRADITION**

   More than one time, tradition was mentioned, above all when the territory as well as the *Tourism Product,* are able to offer typical or folk products – when a tourist goes to the beach, he/she wants sunglasses, umbrellas and deckchair; when a tourist visits Alps, he/she wants to eat mushroom.

3.2.2 View from the Client's side

As we mentioned before, we have created an algorithm enabling to scan the web site in order to search for the occurrences of given keywords. We will present a potential example of clients' requests using keywords.

In this experiment the keyword employed is "*Christmas*".

The experiment is divided in two phases:

1. In the first phase, we ran *Wordrelations* inside Google APIs stopping at the first level of crawling;

2. In the second phase, we ran an adaptation of *Crawler* in order to find word occurrences in a given web site.

   Since *Crawler* requires two parameters – a given keyword and a URL – we input the potential keyword related to tourism, "*Christmas*", and the URL (the American web site https://www.timeanddate.com/holidays/us/christmas-day) related to "*Christmas*", in order to find the occurrences of the given keyword.

The word "*Christmas*" occurred 128 times in *Crawler*.

This example shows the possible utilization of the algorithm: starting from given keywords we can create a list of the most frequent, save the results and create links among the given keywords.

	Keywords	*Wordrelations*	*Crawler*
1	**2017**	7	674
2	**america**	21	50
3	**beach**	0	2
4	**california**	2	2
5	**car**	311	606
6	**cruise**	0	0
7	**day**	1314	12193
8	**family**	96	152
9	**flight**	1	20
10	**food**	51	18
11	**friends**	24	119
12	**holiday**	579	4424
13	**italy**	2	286

	Keywords	*Wordrelations*	*Crawler*
14	**journey**	1	3
15	**love**	84	105
16	**mountain**	0	4
17	**plane**	2	75
18	**santa**	35	2
19	**sharm**	0	0
20	**ship**	25	27
21	**ski**	54	50
22	**snow**	27	6
23	**travel**	36	204
24	**tree**	214	20
25	**trento**	1	1
26	**usa**	59	393

Table 15 ~ Client side example

4. A new proposal

In this chapter, we present the indicators and the parameters of the weighted average of the *Appeal*.

The chapter is divided into three parts:

1. In the first we study the indicators adopted to analyze the final *Appeal*;

2. In the second part we show the structure of the database which is used in the project;

3. The third part describes the algorithm that is used to create the weighted average of the *Appeal*.

The three parts describe our program to be used for data retrieval and *Appeal* definition.

We are also able to use the project as a benchmark structure: in fact, we can compare two *Tourism Products* with similar

features. The algorithm shows where a *Tourism Product* is better performing than the other one.

4.1 Indicators definition

We will analyze the indicators of the *Appeal*.

We found these ten indicators during the training in CISET department: they are the results obtained from the four interviews and the eight pre-existent services.

4.1.1 *City*

It is an important indicator because it considers the site in which the *Tourism Product* is located. The results are different whether we deal with a city or a town.

Famous museums of European capitals – British Museum in London, or Museo degli Uffizi in Florence – have a greater number of visitors every day than a small town in the suburbs.

In the algorithm that defines the *Appeal*, we use four distinctions.

Dimension of City	***City* value**
Capital City	50
Chief	30
Town	15
Village	5

Table 16 ~ City division

4.1.2 *Typology*

In Chapter one we analyzed different kinds of typology: these distinctions are included in our database when we specify the *Tourism Product*.

When we compare two *Tourism Products*, the algorithm checks if the *Typology* of both products are the same. We can not compare two *Tourism Products* that are not of the same *Typology* – e.g. we do not compare a *Service in Loco* with a *Holiday Maker*.

When we try to compare two different *Typologies* of two *Tourism Products,* the algorithm breaks showing a message error.

4.1.3 *Visitors number*

To get an idea of the *Tourism Product* progress, we have to focus on the number of visitors.

This indicator is important when we have to evaluate some features of the tourism improvement. It is possible to retrieve it from the official statistics and Open Data.

4.1.4 *Period*

The period shows the time in which the *Tourism Product* is open.

In the algorithm, *Period* corresponds to the months in which the *Tourism Product* is open – e.g. if a *Tourism Product* is

open to tourists four months a year, the *Period* value is equal to 4.

4.1.5 *Evaluation number*

Evaluation number is the number of the reviews of the *Tourism Product* which we are analyzing.

In the algorithm, this indicator is not used alone, but in relation with the *Evaluation quality*.

4.1.6 *Evaluation quality*

Evaluation quality is strictly linked to the previous indicator.

In the algorithm, this indicator is the arithmetic mean of all the evaluations.

In the *Table 17* we show the five possible evaluation criteria.

Significance	***Evaluation* value**
Very bad	1
Low	2
Sufficient	3
Good	4
Excellent	5

Table 17 ~ Evaluation value

4.1.7 *Area*

Area is the surface area of the *Tourism Product*. The value is given by Open Data or statistic information.

In the algorithm, this indicator is used to calculate the *Capacity*[23].

[23] *Capacity* is one of the eight parameters of the algorithm. The eight parameters will be described in paragraph 4.3.

4.1.8 *Cost*

The indicator shows the total costs of the *Tourism Product* – e. g. tickets and food prices.

Cost is the arithmetic average of the *Tourism Product* prices.

4.1.9 *How to arrive*

The indicator shows the transport available and used to reach the *Tourism Product*.

If the *Tourism Product* is *Transport typology*, the indicator is *False.* Otherwise, the indicator is *True* and shows the frequencies of the different pubblic transports.

4.1.10 *PageRank Results*

Using the *PageRank* algorithm, we get the rank of the *Tourism Product*.

In the algorithm, *PageRank Results* is a number that corresponds to the percentage of frequency in the World Wide Web.

4.2 Database structure

The system we have created needs a database. In the database we save the *Tourism Products*, the *Product Managers* and the *Clients* that are logged.

Figure 7 shows the relational model of the database.

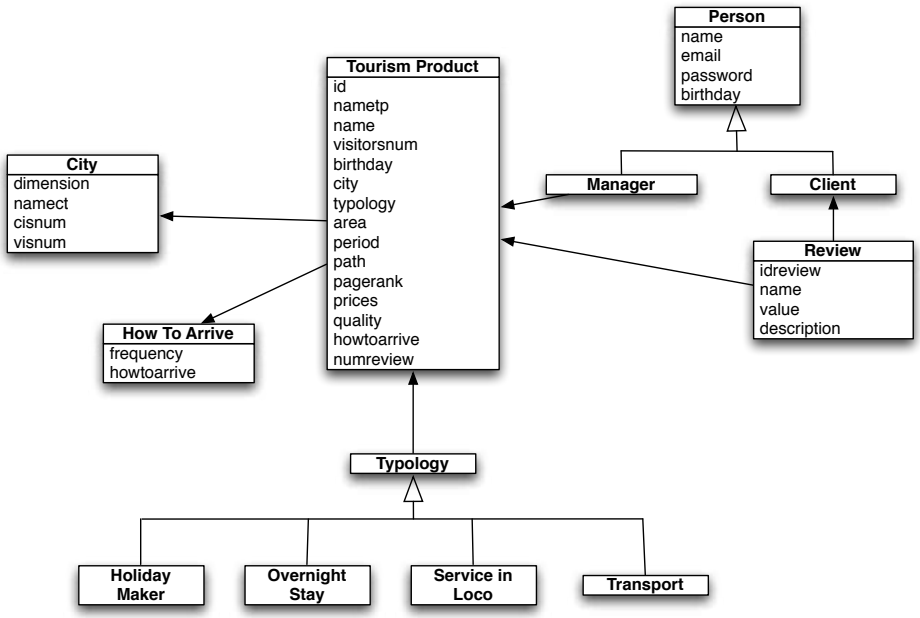

Figure 7 ~ Database Structure

The database is composed of twelve classes – we will study each class in detail.

4.2.1 *Person*

The class specifies the *Person*: it is a superclass, with determined attributes. In *Person* we find the *Product Manager* and the *Client*: they are saved when we they log in the system.

ATTRIBUTES

name	String type. It is univocal for each person logged in. There is a check if the name does not exist, if not, the program asks for a new name.
email	String type. The personal email is used to send and receive information.

ATTRIBUTES

password	String type. When there is a login, the password is sent to the program with the hashed password (SHA256 with a salt): in this way we have an higher security control[24].
birthday	Date type. It is built in the following schema: mm/dd/yyyy where mm means month, dd day and the last one is the year.

Table 18 ~ Person class

[24] In order to avoid that a password could be sniffed while flowing an unprotected network, the client sends a hashed password, the server applies the salt and hashes the password again. Then the server checks whether the password is the same as the one saved in the database [14].

4.2.2 *Client*

It is a subclass that extends the *Person* class.

Clients who sign up into the service are saved in *Client* class.

4.2.3 *Manager*

This is the other subclass that extends Person class.

Product Managers are saved in *Manager class*.

4.2.4 *Review*

In *Review* there are all the reviews of the relative *Tourism Product.*

ATTRIBUTES

idreview	Integer type. It defines the *Review*. Each *Review* has an univoque ID.
name	Client type. It is a reference to *Client* class, because every review has a client.
value	Integer type. It can change from one to five.
description	Text type. It is the text in which there is the review.

Table 19 ~ Review class

4.2.5 *Tourism Product*

This class is complex because it contains the higher number of indicators used in the weighted average. The class defines the *Tourism Product* that we are analyzing.

Product Manager fills an online form which comprehends the following attributes.

ATTRIBUTES

id	Integer value. It defines the *Tourism Product*. Each *Tourism Product* has an univoque ID.
nametp	String type. Each touristic activity has a name: to avoid that there could be possible conflicts with homonym, the server checks whether the name has not been already saved.

ATTRIBUTES

name	*Manager* type. It is the *Product Manager.*
visitornum	Interger number. Visitor presences. This value can change during the activity opening.
birthday	Date type. When the *Tourism Product* has been started.
city	*City* type. The name of the city in which the activity is located in.
typology	*Typology* tpe. What kind of typology.
area	Integer type. It defines the area of the *Tourism Product* (m^2).

ATTRIBUTES

period	Integer type. It defines the opening period. The number indicates the months (range 1 and 12).
path	Boolean type. The value verifies whether the *Tourism Product* can be reached via pubblic transport. If it is TRUE the table named *How To Arrive* is created and contains the attribute *frequency* and *howtoarrive*. In contrast, if the Boolean is FALSE, the relative table is NULL.
pagerank	Integer type. It is the result of the *PageRank* algorithm. The result belongs to a range 0-100.
prices	Integer type. It is the arithmetic mean of all the prices of the *Tourism Product.*

ATTRIBUTES

quality	Integer type. This attribute is the arithmetic average of all the *Evaluation quality* of the *Tourism Product.* The value belongs to the range 1-5.
howtoarrive	*How To Arrive* type. It refers to the class *How To Arrive.*
numreview	Integer type. It is the total number of the reviews.

Table 20 ~ Tourism Product class

4.2.6 *How To Arrive*

This class indicates whether it is possible to reach the *Tourism Product* via public transport.

ATTRIBUTES

frequency	Integer type. Frequency is the result from Open Data and it shows the timetables of each public transport which allows to get to the *Tourism Product*. Three different ranges: 1. First range → from 1 min to 1 hour time, *frequency* = 3; 2. Second range → from 1 hour to 3 means per day, *frequency* = 2; 3. Third range → over 3 pubblic means per day, *frequency* = 1.
howtoarrive	Integer type. The attribute shows the trasports available: 1. PLANE, *howtoarrive* = 2; 2. TRAIN or BOAT, *howtoarrive* = 3.

Table 21 ~ How To Arrive class

4.2.7 *City*

The class indicates the city in which the *Tourism Product* is located. The *Tourism Product* is generally located within a *City* which has several other *Tourism Product*.

ATTRIBUTES

dimension	Integer type. It indicates the type of the *City*. It is classified into four values (see paragraph 4.2.2).
namect	String type. It is the name of the *City*.
cisnum	Integer type. It is the number of citizens.
visnum	Integer type. It is the number of the visitors.

Table 22 ~ City class

4.2.8 *Typology*

It is a superclass. It indicates the *Typology* of the *Tourism Product* through four subclasses: *Holiday Maker*, *Overnight Stay*, *Service In Loco*, *Transport*.

4.2.9 *Holiday Maker*

It is the first subclass of *Typology*.

In *Holiday Maker* we insert the *Tourism Product* that corresponds to this definition.

4.2.10 *Overnight Stay*

It is the second subclass of *Typology*.

In *Overnigh Stay* we insert the *Tourism Product* that corresponds to this definition.

4.2.11 *Service In Loco*

It is the third subclass of *Typology*.

In *Service In Loco* we insert the *Tourism Product* that corresponds to this definition.

4.2.12 *Transport*

It is the fourth subclass of *Typology*.

In *Transport* we insert the *Tourism Product* that corresponds to this definition.

4.3 The *Appeal* definition

In order to obtain the *Appeal* weighted average, we insert eight variables (here indicated with capital letters).

	Name	Description
1	TYPOLOGY	TYPOLOGY indicates the typology of the *Tourism Product*. When comparing two *Tourism Products* of different typologies, the program breaks immediately. It is used to check the comparison.
2	CITY	It indicates the city in which the *Tourism Product* is located in.
3	PERIOD	It indicates the opening months.

	Name	Description
4	PRICES	PRICES has different values, related to the *Typology*: • If the *Tourism Product* is a *Service In Loco*, PRICES is equal to $\frac{prices\ of\ the\ Tourism\ Product}{PERIOD}$ • If the *Tourism Product* is a *Holiday Maker*, PRICES is equal to $\frac{Area}{\frac{prices\ of\ the\ Tourism\ Product}{PERIOD}}$ • If the *Tourism Product* is an *Overnight Stay*, PRICES is equal to $\frac{prices\ of\ the\ Tourism\ Product}{PERIOD}$ • If the *Tourism Product* is a *Transport*, PRICES is equal to $\frac{prices\ of\ the\ Tourism\ Product}{visitors\ number + number\ of\ the\ citizen}$

	Name	Description
5	CAPACITY	The maximum amount of people that the *Tourism Product* can contain. CAPACITY is given by $\frac{Area}{Visitor\ Number}$
6	EVALUATION	EVALUATION is given by $\frac{Evaluation\ Number}{Arithmetic\ average\ of\ quality}$
7	WAY	WAY is the results of the sum of the frequency and the transport used to reach the *Tourism Product*. It is given by $frequency + howtoarrive$
8	PAGERANK	PAGERANK is the result of the *PageRank* algorithm

Table 23 ~ Appeal definition

The weighted average is given by:

$$\frac{CITY + PERIOD + PRICES + CAPACITY + EVALUATION + WAY + PAGERANK}{7}$$

Table 24 ~ Weighted Average of Appeal

In *Code 1* we show the portion of the code that quantifies the *Appeal* of a *Tourism Product*.

```
# Appeal returns an int
    # Appeal is the value of the Tourism Product
    def Appeal(tourism):

        # Parameters used in the weighted average
        # CITY
        ct = tourism.getcity().getbig()
        # CAPACITY
        su = tourism.getsurface()
        vs = tourism.getvisitornum()
        capacity = su/vs
        # EVALUATION
        ev = tourism.getreview()
        qu = tourism.getquality()
        evaluation = ev/qu
        # PAGERANK
        pa = tourism.getpagerank()
        # WAY
        wy = tourism.gethowtoarrive()
        way = 0
        # Check if the Tourism Product has the parameter
        if wy == False:
            way
        elif wy == True:
            # WAY
            w1 = tourism.gethowtoarrive().getfrequency()
            w2 = tourism.gethowtoarrive().gethowtoarrive()
```

```
28            way = w1 + w2
29        # PRICES
30        pr = tourism.getprices()
31        # PERIOD
32        pe = tourism.getperiod()
33        # We differentiate the different typologies
34        # because the prices change
35        if tourism.gettypology() == Serviceinloco:
36            prices = pr / pe
37        elif tourism.gettypology() == Holidaymaker:
38            prices = (su / pr) / pe
39        elif tourism.gettypology() == Overnightstay:
40            prices = pr / pe
41        else:
42            prices = pr / (tourism.getvisitornum() +
43                      tourism.getcity().getcinum())
44
45        # We take seven parameters to have the final Appeal
46        # “ct” capital/chief/town/village
47        # “pe” is the opening months
48        # “prices” is the arithmetic average of the prices
49        # “capacity” is the relationship between
50        # “surface area” and “number of visitors”
51        # “evaluation” is the relationship between
52        # “number of evaluation” and
53        # “average of the quality”
54        # “way” is the result
55        # of “frequency” and “transport”
56        # “pa” is the result of PageRank
57        appeal = (
58                    ct + pe + prices +
```

```
59                  capacity + evaluation + way + pa) / 7
60
61        print("The final Appeal of " +
62              tourism.getnametp() +" is:")
63        print(appeal)
64 print('\n')
```

Code 1 ~ Appeal

The output is the following.

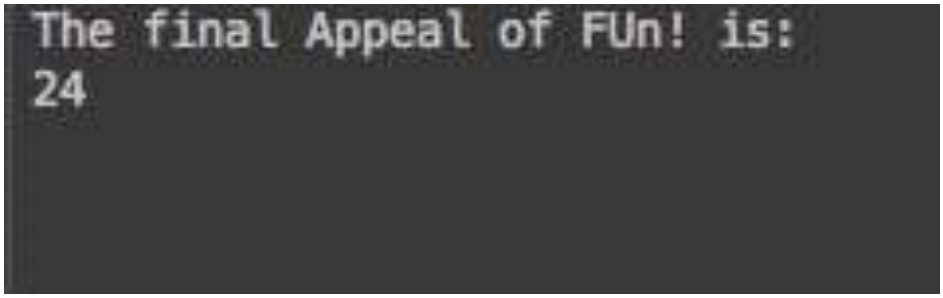

Figure 8 ~ Appeal output

In the Appendix D we present the whole code with the database and three potential examples of *Tourism Product.*

4.3.1 Comparison of two potential *Tourism Products*

We use the program as a benchmark that compares two different *Tourism Products* of the same *Typology*.

The *Product Manager* can have the results of the progress of his/her *Tourism Product* and the concurrency's: in this way, he/she has the possibility to improve what the *Tourism Product* is lacking with respect the other *Tourism Product*.

The program compares each component of the *Appeal* weighted mean and shows where a *Tourism Product* is better than the other one.

The features are the same of the *Appeal* definition.

The *Code 2* shows the portion of code that compares two *Tourism Products*.

```
# We compare two Tourism Products with the same Typology
    def Compare(t1, t2):

        Appeal(t1)
        Appeal(t2)
        # Parameters of the first TP
        # CITY t1
        ct1 = t1.getcity().getbig()
        # PERIOD t1
        pe1 = t1.getperiod()
        # CAPACITY t1
        su1 = t1.getsurface()
        vs1 = t1.getvisitornum()
        capacity1 = su1/vs1
        # PRICES t1
        pr1 = t1.getprices()
        # We differentiate the different typologies,
        # first Tourism Product
        if t1.gettypology() == Serviceinloco:
            prices1 = pr1 / pe1
        elif t1.gettypology() == Holidaymaker:
            prices1 = (su1 / pr1) / pe1
        elif t1.gettypology() == Overnightstay:
            prices1 = pr1 / pe1
        else:
            prices1 = pr1 / (
                t1.getvisitornum() +
                t1.getcity().getcinum()
                )
        # EVALUATION t1
        ev1 = t1.getreview()
```

```
32          qu1 = t1.getquality()
33          evaluation1 = ev1/qu1
34          # WAY t1
35          wy1 = t1.gethowtoarrive()
36          # Check if the first Tourism Product
37          # has the parameter
38          way1 = 0
39          if wy1 == False:
40              way1
41          elif wy1 == True:
42              w1t1 = t1.gethowtoarrive().getfrequency()
43              w2t1 = t1.gethowtoarrive().gethowtoarrive()
44              way1 = w1t1 + w2t1
45          # PAGERANK t1
46          pa1 = t1.getpagerank()
47
48          # Parameters of the second TP
49          # CITY t2
50          ct2 = t2.getcity().getbig()
51          # PERIOD t2
52          pe2 = t2.getperiod()
53          # CAPACITY t1
54          su2 = t2.getsurface()
55          vs2 = t2.getvisitornum()
56          capacity2 = su2/vs2
57          # PRICES t2
58          pr2 = t2.getprices()
59          # We differentiate the different typologies,
60          # second Tourism Product
61          if t2.gettypology() == Serviceinloco:
62              prices2 = pr2 / pe2
```

```
63          elif t2.gettypology() == Holidaymaker:
64              prices2 = (su2 / pr2) / pe2
65          elif t2.gettypology() == Overnightstay:
66              prices2 = pr2 / pe2
67          else:
68              prices2 = pr2 / (
69                  t2.getvisitornum() +
70                  t2.getcity().getcinum()
71                  )
72          # EVALUATION t2
73          ev2 = t2.getreview()
74          qu2 = t2.getquality()
75          evaluation2 = ev2/qu2
76          # WAY t2
77          wy2 = t2.gethowtoarrive()
78          # Check if the second Tourism Product
79          # has the parameter
80          way2 = 0
81          if wy2 == False:
82              way2
83          elif wy2 == True:
84              w1t2 = t2.gethowtoarrive().getfrequency()
85              w2t2 = t2.gethowtoarrive().gethowtoarrive()
86              way2 = w1t2 + w2t2
87          # PAGERANK t2
88          pa2 = t2.getpagerank()
89
90          # We compare the parameters of both TPs
91          while True:
92
93              # First control,
```

```
94              # if the TPs are different,
95              # we can not compare them
96              if t1.gettypology() != t2.gettypology():
97                  print("We can't compare them!")
98                  print('\n')
99                  break
100
101                  # We compare the city
102                  # where the TPs are located in
103                  if ct1 > ct2:
104                      print("CITY")
105                      print(
106                          "The City "
107                          + t1.getnametp()
108                          + " is bigger"
109                          )
110                      print('\n')
111                  elif ct2 > ct1:
112                      print("CITY")
113                      print(
114                          "The City "
115                          + t2.getnametp() +
116                          " is bigger"
117                          )
118                      print('\n')
119                  elif ct1 == ct2:
120                      print("CITY")
121                      print(
122                            "The cities are big both"
123                            )
124                      print('\n')
```

```
125
126                 # We compare the period of opening
127                 if pe1 > pe2:
128                     print("PERIOD")
129                     print(
130                         "The opening months of "
131                         + t1.getnametp() +
132                         " are bigger then "
133                         + t2.getnametp()
134                         )
135                     print('\n')
136                 elif pe2 > pe1:
137                     print("PERIOD")
138                     print(
139                         "The opening months of "
140                         + t2.getnametp() +
141                         " are bigger then "
142                         + t1.getnametp()
143                         )
144                     print('\n')
145                 else:
146                     print("PERIOD")
147                     print(
148                           "The opening months "
149                           "of the TPs are equal"
150                           )
151                     print('\n')
152
153                 # We compare the arithmetic
154                 # averages of the prices
155                 if prices1 > prices2:
```

```
156                     print("PRICES")
157                     print(
158                         t1.getnametp() +
159                         " is more expensive than "
160                         + t2.getnametp()
161                         )
162                     print('\n')
163                 elif prices2 > prices1:
164                     print("PRICES")
165                     print(
166                         t2.getnametp() +
167                         " is more expensive than "
168                         + t1.getnametp()
169                         )
170                     print('\n')
171                 else:
172                     print("PRICES")
173                     print(
174                           "The prices are equal"
175                           )
176                     print('\n')
177
178                 # We compare the capacities
179                 if capacity1 > capacity2:
180                     print("CAPACITY")
181                     print(
182                           "The capacity of "
183                           + t1.getnametp() +
184                           " is bigger than "
185                           + t2.getnametp()
186                       )
```

```
187                     print('\n')
188                 elif capacity2 > capacity1:
189                     print("CAPACITY")
190                     print(
191                         "The capacity of "
192                         + t2.getnametp() +
193                         " is bigger than " +
194                         t1.getnametp()
195                         )
196                     print('\n')
197                 else:
198                     print("CAPACITY")
199                     print(
200                           "The capacities are equal"
201                           )
202                     print('\n')
203
204                 # We compare the evaluations
205                 if evaluation1 > evaluation2:
206                     print("EVALUATION")
207                     print(
208                           "The evaluation of "
209                           + t1.getnametp() +
210                           " is bigger than " +
211                           t2.getnametp()
212                          )
213                     print('\n')
214                 elif evaluation2 > evaluation1:
215                     print("EVALUATION")
216                     print(
217                          "The evaluation of " +
```

```
218                             t2.getnametp() +
219                             " is bigger than " +
220                             t1.getnametp()
221                         )
222                     print('\n')
223                 else:
224                     print("EVALUATION")
225                     print(
226                         "The number evaluations are equal"
227                         )
228                     print('\n')
229 
230                 # We compare way parameter
231                 if way1 > way2:
232                     print("WAY")
233                     print(
234                         t1.getnametp() +
235                         " is easier to reach"
236                         )
237                     print('\n')
238                 elif way2 > way1:
239                     print("WAY")
240                     print(
241                         t2.getnametp() +
242                         " is easier to reach"
243                         )
244                     print('\n')
245                 else:
246                     print("WAY")
247                     print(
248                         "The TPs have the same way to reach"
```

```
249                         )
250                     print('\n')
251
252                 # We compare the PageRank results
253                 if pa1 > pa2:
254                     print("PAGERANK")
255                     print(
256                         "The PageRank result is bigger in "+
257                         t1.getnametp()
258                         )
259                     print('\n')
260                 elif pa2 > pa1:
261                     print("PAGERANK")
262                     print(
263                         "The PageRank result is bigger in "+
264                         t2.getnametp()
265                         )
266                     print('\n')
267                 else:
268                     print("PAGERANK")
269                     print("The PageRank
270                           results are the same")
271                     print('\n')
272
273                 break
```

Code 2 ~ Compare

In Appendix D we present the whole code with the database and three potential examples of *Tourism Products.*

The program checks if the comparison is possible, otherwise, it breaks with an error message.

Figure 9 and *Figure 10* show the relative outputs.

```
The final Appeal of JetMarket is:
102


The final Appeal of FUn! is:
24


CITY
The City FUn! is bigger


PERIOD
The opening months of JetMarket are bigger then FUn!


PRICES
JetMarket is more expensive than FUn!


CAPACITY
The capacities are equal


EVALUATION
The evaluation of JetMarket is bigger than FUn!


WAY
The TPs have the same way to reach


PAGERANK
The PageRank result is bigger in FUn!
```

Figure 9 ~ Compare result

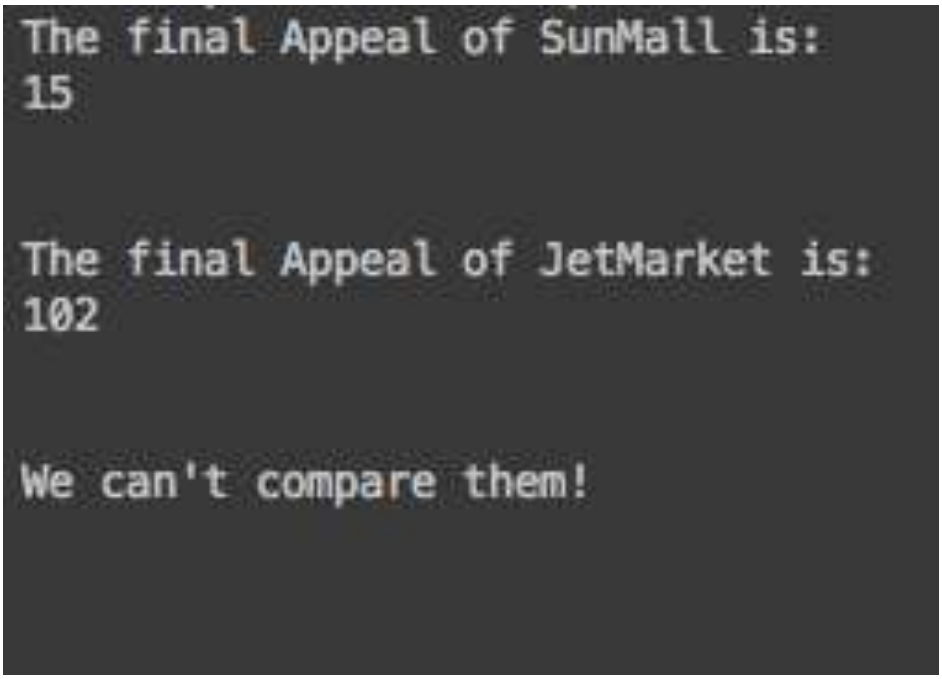

Figure 10 ~ Compare -Error message

5. Conclusions

In the thesis we presented the *Appeal* in the *Destination Monitor*.

We wanted to realize a prototype of the program that retrieves the data from the Web and defines the *Appeal*. We chose two simples languages, similar to pseudo code, that could be immediately understood. The database is quite small, because we wanted to emphasize the final results.

We are aware that the program has some limitations, however it is open to improvement and further research, both optimizing hardware and better performancing programs, e. g:

- usin objective languages (like C# or JAVA) which are more performative when using a big storage;

- when using Big Data of information, the database can be stored in high performance disks (with a lower latency when accessing the memory);

- since *Crawler*, *PageRank* and *Wordrelations* work on little portion of the Web, they can be optimized by using different data structure or changing the hardware components.

Some future works will focus on optimizating the hardware, or the better performance of the program and the access disks in which the data are saved.

Using appropriate indicator changes we can apply the project to different environments. For example, we can define the *Appeal* of a Movie (*Movie Appeal)*: it will be possible to analyze the feedbacks from social networks and other tools present in the Web; then it will be possible to analyze which are the indicators, hence define the components in the average weight to calculate the final *Appeal*.

Glossary

API: Application Programming Interface. It is a set of definitions, protocols and tools for building software and application. It specificies how software components should interact (Collins).

[Chap. 1, par. 1.2, page 20]

Appeal: The power of a Tourism Product to attract please, stimulate, or interest (Collins).

[Chap. 2, par. 2.1.1, page 27]

Array: A regular data structure in which individual elements may be located by references to one or more integer index variables. (Collins)

[App. A, page 117]

[App. B, page 121]

Benchmark: A criterion by which to measure something; standard; reference point. (Collins)

[Chap. 4, par. 4.3.1, page 87]

Big Data: It is a huge amount of information that can only be computed by special computers.

[Chap. 5, page 101]

Complexity: The time complexity of an algorithm qunatifies the amount of time taken by the algorithm itself to run as a function.

[Chap. 5, page 101]

Crawler: A computer program that is capable of performing recursive searches on the Internet. (Collins)

[Chap. 2, par. 2.1.2 page 28]

[App. A, page 117]

Dashboard: User interface that organizes and presents information in a way that is easy to read.

[Chap. 2, par. 1.2, page 19]

Destination Monitor: System that monitors the destination using indicators.

[Chap. 1, page 7]

Graph: A draw that explains the relation between certain elements by means of series of dots and lines.

[Chap. 2, par. 2.1.3 page 32]

Google API: It is a set of application programming interfaces developed by Google which allows communication with Google Services and their integration to other services. (Google)

[Chap. 2, par. 2.1.4 page 35]

Indicator: something that provides an indication. (Collins)

[Chap. 2, par. 2.2, page 38]

[Chap. 4, par. 4.1, page 62]

Open Data: Data that are available to everyone to use and repiblishes as they wish, without restrictions of copyright.

[Chap. 1, par. 1.2, page 17]

[Chap. 2, page 23]

PageRank: Algorithm used to rank web sites in search engine results.

[Chap. 2, par. 2.1.3, page 32]

Prestige: The power to influence.

[Chap. 2, par. 2.1.3, page 33]

[App. B, page 121]

Product Manager: Person who is the manager of the *Tourism Product*.

Rank: Importance of a determinate page.

[Chap. 2, par. 2.13, page 33]

Relational Model: It is the representation of the organization of data into collections of two-dimensional tables called relations.

[Chap. 4, par. 4.2, page 68]

Salt: It is a random number used as an additional input to a one-way-hash password.

[Chap. 4, par 4.2.1, page 70]

SHA256: Secure Hash Algorithm, set of cryptographic hash functions.

[Chap. 4, par. 4.2.1, page 70]

Singleton: It is a design pattern that restricts the instantiation of a object.

[App. C, page 128]

Superclass: It is a class from which other classes are derived.

[Chap. 4, par. 4.2, page 69]

Tourism Product: It is any product in the field of tourism.

UML: Unified Modeling Language, it shows the actions of the various actors present in a determinate project.

[Chap. 2, par. 2.1, page 25]

[Chap. 3, par. 3.1, page 48]

[Chap. 3, par. 3.1, page 50]

URL: Uniform Resource Locator, string that addresses to a determinate web site.

[Chap. 2, par. 2.1.2, page 28]

[Chap. 2, par. 2.1.4 , page 35]

[App. A, page 117]

[App. C, page 127]

Web graph: It describes the directed links between pages in the World Wide Web.

[Chap. 2, par 2.13, page 32]

References

BOOKS AND PAPERS

[1] Celotto E., Minghetti V., *Destination Web reputation: Combining explicit and implicit popularity to build an integrated monitoring system*, CISET – Ca' Foscari University Venice, 2015.

[2] Cerato M., Manente M., *Destination Management: the conceptual framework*, CISET – Ca' Foscari University Venice, 2000.

[3] Minghetti V., *Il turismo come risorsa strategica: concetti, indicatori, attori e tendenze,* CISET – Ca' Foscari University Venice, 2016.

[4] Manente M., *Destination management and economic background: defining and monitoring local toirist destinations,* CISET – Ca' Foscari University Venice, 2008.

[5] Chiarullo L., Maggiore M., *L'innovazione nella gestione delle informazioni turistiche in Puglia: il caso SIR – Tur,* 2013.

[6] InnovaPuglia, *SPOT: Sistema Puglia per l'Osservatorio Turistico – Innovazione e semplificazione nella gestione delle informazioni turistiche,* 2013.

[7] InnovaPuglia, *SPOT: Sistema Puglia per l'Osservatorio Turistico – Modalità Base – Specifiche del sistema,* 19/12/2012.

[8] InnovaPuglia, *SPOT: Sistema Puglia per l'Osservatorio Turistico – Modalità Base – Manuale d'uso del sistema,* 19/12/2012.

[9] Giunta Regionale Emilia-Romagna, *Linee giuda relative al riutilizzo e messa a disposizione in Open Data dei dati pubblici dell'amministrazione regionale,* 2012.

[10] Ferrarese C., Schiavon G., *PageRank project,* a. y. 2015-2016

[11] Collins, *Concise English Dictionary*, Zanichelli, eight edition 2012

www.collinslanguage.com

[12] K. Brann; G. Giola; D. Kirby; W. Richtmyer, *25 Years of Venice Knowledge Online*, WPI Interactive Qualifying Project, Venice Project Center, December 2013.

WEB SITES

[13] Orlando S., Data and Web Mining, http://www.dsi.unive.it/~dm/, a.y. 2014/2015.

[14] Focardi R. Security, https://secgroup.dais.unive.it/teaching/security-course/identification/

[15] TripAdvisor, https://www.tripadvisor.it/PressCenter-c6-About_Us.html

[16] PaesiOnLine, http://www.paesionline.info/company.asp?domain=.it

[17] CastelliExperience, http://www.castelliexperience.it/?page_id=3394

[18] Trivago, http://company.trivago.it

[19] Expedia.it, https://www.expedia.it/p/support/chi-siamo

[20] GroupOn, https://www.groupon.it/about-us

[21] Booking.com, http://www.booking.com/content/about.it.html?label=gen173nr-1FCAEoggJCAlhYSDNiBW5vcmVmaHGIAQGYARS4AQfIAQ3YAQHoAQH4AQuoAgM;sid=181bbe04226965633091bf7d303b444b

[22] Venere.com, http://it.venere.com/page/about-us/

[23] TravelAppeal, https://www.travelappeal.com/it/

[24] FourTourism, http://www.fourtourism.it/chi-siamo/,

http://www.fourtourism.it/reputation/

[25] Evols,

http://www.evols.it/it/#/chi-siamo/

[26] Reputation Manager,

http://www.reputazioneonline.it/chi-siamo

[27] Venice Project Center,

https://github.com/sunng87/heatcanvas

http://veniceprojectcenter.org

[28] Trentino Region,

http://www.trentinomarketing.org/it/cosa-facciamo/progetti-di-sistema/h-benchmark/

http://www.develongroup.com

http://www.trustyou.com

http://hotelexecutive.com/author/800

[29] Puglia Region,

http://www.agenziapugliapromozione.it/portal/spot

[30] Piedmont Region,
http://www.dati.piemonte.it

[31]Emilia Romagna Region,
http://digitale.regione.emilia-romagna.it/dati/temi/accesso-ai-dati
http://digitale.regione.emilia-romagna.it/piter

[32] Lombardy Region,
https://www.dati.lombardia.it
https://socrata.com/case-study/lombardia-open-data-designed-for-growth/
https://socrata.com/case-study/lombardia-using-open-data-drive-performance-accountability-change-lives/
https://www.youtube.com/watch?v=Lyew8C6tcsk

Appendix A

Appendix A refers to paragraph 2.1.2.

Crawler code

Crawler needs two elements as input: the first one is the URL in which it searches for the keywords, the second is an array of keywords.

The code is written in Python language.

```
1 import requests
2 import re
3 import urlparse
4
5  # In this example we are trying to collect
6  # the occurrences
7  # of touristic words
8  # HTML <a> regexp
9  # Matches href="" attribute
10 link_re = re.compile(r'href="(.*?)"')
11
12 # Words list
13 wlist = ['cruise']
14
```

```
15 def crawl(url, maxlevel):
16     # Limit the recursion, we are not downloading
17     # the whole Internet, a crawler goes in depth
18     if(maxlevel == 0):
19         return [0,0]
20 
21     # Get the web page
22     req = requests.get(url)
23     result = [0,0]
24 
25     # Check if successful
26     if(req.status_code != 200):
27         return [0,0]
28 
29     # Find and follow all the links
30     links = link_re.findall(req.text)
31     for link in links:
32         # Get an absolute URL for a link
33         link = urlparse.urljoin(url, link)
34         temp = crawl(link, maxlevel - 1)
35         for i, v in enumerate(temp):
36             result[i] += v
37 
38     # Find all occurrences on current page
39     count = 0
40     index = 0
41     for word in wlist:
42         while True:
43             index = req.text.find(word, index + len(word))
44             if index == -1:
45                 index = 0
```

```
46                 count += 1
47                 break
48             else:
49                 result[count] += 1
50
51
52     return result
53
54 # We insert the URL in the parameter
55 def main():
56     occurences = \
57         crawl(
58             'https://www.timeanddate.com/'
59             'holidays/us/christmas-day'
60             , 2)
61
62     print("Results:")
63     for i,word in enumerate(wlist):
64         print("Word: " + word + " --> "
65               + str(occurences[i]) + " occurences")
66
67 if __name__ == "__main__":
68 main()
```

Code 3 ~ Crawler

Appendix B

Appendix B refers to paragraph 2.1.3.

PageRank code

The input is a txt file in which there are two columns: the first represents the outgoing link from one node of the web graph, the second column is the ingoing link to another node. This algorithm summarizes the graph of connections between nodes. It is written in C language.

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <math.h>
4
5  int main() {
6      // number of nodes
7      int n = 100;
8      // indexes
9      int i = 0;
10     int j = 0;
11     // input file
12     FILE *fp;
```

```
13    // matrix
14    float matrix[n][n];
15    // vector of outgoing links
16    int out_link[n];
17    // transposed matrix
18    float t_matrix[n][n];
19    // p vector
20    float p[n];
21    // new p vector
22    float p_new[n];
23    // iteration counter
24    int k = 0;
25    float error = 0;
26    int looping = 1;
27    // damping factor
28    float d = 0.85;
29
30    // initialization of the matrix
31    for ( i=0; i<n; i++) {
32        for( j=0; j<n; j++){
33            matrix[i][j]=0;
34        }
35    }
36
37    // initialization of the transposed matrix
38    for ( i=0; i<n; i++) {
39        for( j=0; j<n; j++){
40             t_matrix[i][j]=0;
41        }
42    }
43
```

```
44    // initializing the outlinks vector
45    for (i=0; i<n; i++){
46        out_link[i]=0;
47    }
48
49    // initializing the p vector
50    for (i=0; i<n; i++){
51        p[i]=1.0/n;
52    }
53
54    printf("File: \n");
55    // opening file
56    fp = fopen("file100.txt", "r");
57    while (!feof(fp)) {
58        // reading numbers from file
59        fscanf(fp, "%d %d\n", &i, &j);
60        i = i;
61        j = j;
62        // putting the edges in the matrix
63        matrix[i][j]= 1;
64    }
65
66    // counting the outlinks
67    for (i=0; i<n; i++){
68        for (j=0; j<n; j++){
69            if (matrix[i][j] != 0){
70                out_link[i] =
71                out_link[i] + 1;
72            }
73        }
74    }
```

```
75
76    // normalizing the dangling nodes' rows
77    for (i=0; i<n; i++){
78        if (out_link[i] == 0){
79           for (j=0; j<n; j++){
80                 matrix[i][j]=1.0/n;
81           }
82        }
83        else{
84            // normalizing the values of the rows
85            for (j=0; j<n; j++){
86                if (matrix[i][j] != 0.0){
87                    matrix[i][j] =
88                        matrix[i][j]/out_link[i];
89                }
90            }
91        }
92    }
93
94    // transpose
95    for (i=0; i<n; i++){
96        for(j=0; j<n; j++){
97            t_matrix[j][i] = matrix[i][j];
98        }
99    }
100
101       // probability vector
102       while (looping){
103           // initialising the new p vector
104           for (i=0; i<n; i++){
105               p_new[i]=0;
```

```
106             }
107
108             for (i=0; i<n; i++){
109                 for(j=0; j<n; j++){
110                     p_new[i] =
111                         p_new[i] + t_matrix[i][j] * p[j];
112                 }
113             }
114
115             for (i=0; i<n; i++){
116                 p_new[i] =
117                     (p_new[i]*d) + ((1.0 - d)/n);
118                 }
119
120             error=0;
121
122             // check if we have to stop
123             for (i=0; i<n; i++){
124                 error =
125                     error + (fabsf(p_new[i] - p[i]));
126             }
127
128             if (error < 0.000001){
129                 looping = 0;
130             }
131
132             // updating p
133             for (i=0; i<n; i++){
134                 p[i] = p_new[i];
135             }
136
```

```
137             k = k + 1;
138         }
139
140         printf("Final P Vector:\n");
141         // printing the vector
142         for (i=0; i<n; i++) {
143             printf("%3.3f ", p[i]);
144         }
145
146         printf("");
147
148         return 0;
149
151 }
```

Code 4 ~ PageRank

Appendix C

Appendix C refers to paragraph 2.1.4.

Wordrelations code

The inputs are two arrays. In the first we insert the keywords to be searched on Google, the second is the list of words to be searched within a page.

It is written in Python language.

```
1 import sys
2 import requests
3 from requests.packages.urllib3.exceptions \
4     import InsecureRequestWarning
5 import re
6 import urlparse
7 from googleapiclient.discovery import build
8
9 # This is the list of words to be searched on Google
10 primary_keys = []
11
12 # This is the list of words to be searched within a page
13 related_keys = []
14
```

```
15 # How deep do we want to go?
16 level = 1
17
18
19 class WordRelations:
20     # This class is managed as a Singleton
21     # This static variable represents the
22     # current active instance
23     Instance = None
24
25     # Google API keys
26     GOOGLE_SEARCH_ID = \
27         "011139836630747192430:vubffkynfn0"
28     GOOGLE_API_KEY = \
29         "AIzaSyB7NtXFj4x1zTKahqWojfd-HjXXaoumZAQ"
30
31     # To search deeper we need to find
32     # the links within a page
33     # We match here the href attribute
34     PAGE_LINK = re.compile(r'href="(.*?)"')
35
36     # Singleton instantiation
37     @staticmethod
38     def GetInstance():
39         if WordRelations.Instance == None:
40             Instance = WordRelations()
41
42         return Instance
43
44     # We google a certain word
45     def GoogleSearch(self, word):
```

```
46          # We return a list of URLs
47          urls = []
48          # Google API engine must be initialized
49          google = build(
50              "customsearch",
51              "v1",
52              developerKey = WordRelations.GOOGLE_API_KEY)
53          # GOOGLE IT!
54          results = google.cse().list(
55              q = word,
56              cx = WordRelations.GOOGLE_SEARCH_ID
57          ).execute()
58
59          # For each result found by Google
60          # we extract the URL
61          for item in results['items']:
62              # If a result is on a https site,
63              # the URL is already ok
64              # otherwise, add a http://
65              if item['formattedUrl'][0:5] != "https":
66                  urls.append("http://" +
67                              item['formattedUrl'])
68              else:
69                  urls.append(item['formattedUrl'])
70
71          # Return the results
72          return urls
73
74      # Crawl a web page and find
75      # the related words occurrences
76      def Crawl(self, url, deep):
```

```
77          # Limit the recursion,
78          # we are not downloading the whole Internet
79          if(deep == 0):
80              return [0] * len(related_keys)
81
82          # Download the webpage...
83          # Never verify the certificate
84          # (https can be a pain in the neck...)
85          try:
86              req = requests.get(url, verify = False)
87          except:
88              # If something goes wrong with the connection,
89              # just ignore the web site
90              print(
91                  "Unrecoverable error: connection "
92                  "aborted for website: "
93                  + url)
94              return [0] * len(related_keys)
95
96          result = [0] * len(related_keys)
97
98          # Check if successful
99          if(req.status_code != 200):
100                 return [0] * len(related_keys)
101
102             # Once the page is downloaded,
103             # look for all the links
104             links = WordRelations.PAGE_LINK.
105                                     findall(req.text)
106             # For each link...
107             for link in links:
```

```
108                 # Get an absolute URL
109                 link = urlparse.urljoin(url, link)
110 
111                 if link[0:4] != 'http':
112                     continue
113 
114                 # Get the frequencies in the deeper link
115                 temp = self.Crawl(link, deep - 1)
116                 # Update the current to get the total
117                 for i,v in enumerate(temp):
118                     result[i] += v
119 
120             # Search each related keys in the
121             # page and count
122             # how many times they appear
123             count = 0
124             index = 0
125             for word in related_keys:
126                 print("Checking related word: " + word)
127                 while True:
128                     index = req.text.find(
129                         word, index + len(word))
130                     if index == -1:
131                         index = 0
132                         count += 1
133                         break
134                     else:
135                         result[count] += 1
136 
137             return result
138 
```

```
139         # Search each primary word on Google
140         # For each web site crawl it and find
141         # the related key occurrences
142         def CreateWordsRelationship(self):
143             # The graph we're returning
144             words_graph = []
145 
146             # For each word...
147             for word in primary_keys:
148                 print("Checking key: " + word)
149                 # We google it
150                 google_results = self.GoogleSearch(word)
151                 # Init the results for the current word
152                 temp_results = [0] * len(related_keys)
153                 # For each web site found by Google
154                 for url in google_results:
155                     print("Checking website: " + url)
156                     # Find all occurrences
157                     current_results =
158                                 self.Crawl(url, level)
159                     # When found,
160                     # add them to the previous ones
161                     for i, result \
162                             in enumerate(current_results):
163                         temp_results[i] += result
164 
165                 # Ok, the current word has been checked,
166                 # let's Google the next
167                 words_graph.append(temp_results)
168 
169             return words_graph
```

```
170
171         # Search&Crawl and then save the result
172         # on a file called words_graph.txt
173         def main(self):
174             engine = WordRelations.GetInstance()
175
176             results = engine.CreateWordsRelationship()
177
178             try:
179                 out_file = open("./words_graph.txt", "w")
180                 out_file.write("# Columns header: | ")
181                 for key in related_keys:
182                     out_file.write(key + " | ")
183                 out_file.write("\n# Rows header: | ")
184                 for key in primary_keys:
185                     out_file.write(key + " | ")
186
187                 out_file.write("\n\n")
188
189                 for row in results:
190                     for col in row:
191                         out_file.write(str(col) + "\t")
192                     out_file.write("\n")
193
194                 out_file.close()
195             except IOError as e:
196                 print("FATAL: I/O error: " + e.strerror)
197
198             print(
199                 "\n\nExecution ended. Check file "
200                 "./words_graph.txt for results.")
```

```
201
202
203     # If we are in the main,
204     # we create perform some activities and the let's
205     # find the results!
206     if __name__ == "__main__":
207         requests.packages.urllib3.disable_warnings(
208             InsecureRequestWarning)
209
210         if len(sys.argv) < 3:
211             print("USAGE: ./wordrelations.py "
212                   "<primary_words_list>
213                   <related_words_list>\n")
214             print("Lists are in the form"
215                   " item1,item2,...,itemN\n")
216             print("primary_words_list "
217                   "is the list of words to be googled")
218             print("related_words_list is "
219                   "the list of words "
220                   "to be found for each web site "
221                   "found by google\n")
222             exit(0)
223
224         primary_keys = sys.argv[1].split(',')
225         related_keys = sys.argv[2].split(',')
226
227 WordRelations.GetInstance().main()
```

Code 5 ~ Wordrelations

Appendix D

Appendix D refers to paragraphs 4.3.

Appeal and *Compare* codes

In the following *Code 6*, we show the creation of the twelve classes and the *Appeal* and *Compare* codes.

The program is written in Python language.

```
def main():

    # Person class
    class Person:

        def __init__(
                self, name, email, password, birthday
        ):
            # "name" is a string, it is the person's name
            self.name = name
            # "email" is a string,
            # it is the username used to log in
            self.email = email
            # "password" is a string,
            # it is the password to used to log in
            self.password = password
```

```
17              # "birthday" is a string
18              self.birthday = birthday
19
20          # getname() retrieves the person's name,
21          # and setname() changes the person's name
22          def getname(self):
23              return self.name
24
25          def setname(self, a):
26              self.name = a
27              return self.name
28
29          # getemail() retrieves the person's email,
30          # setemail() changes the person's email
31          def getemail(self):
32              return self.email
33
34          def setemail(self, a):
35              self.email = a
36              return self.email
37
38          # getpwd() retrieves the person's password,
39          # setpwd() changes the person's password
40          def getpwd(self):
41              return self.password
42
43          def setpwd(self, a):
44              self.password = a
45              return self.password
46
47          # getbday() retrieves the person's birthday,
```

```
48          # stebday() changes the person's birthday
49          def getbday(self):
50              return self.birthday
51
52          def setbday(self, a):
53              self.birthday = a
54              return self.birthday
55
56      # Manager class, it is a subclass of Person
57      class Manager(Person):
58
59          def __init__(
60                  self, name, email, password, birthday
61          ):
62              Person.__init__(
63                  self, name, email, password, birthday
64              )
65
66      # Client class, is a subclass of Person
67      class Client(Person):
68
69          def __init__(
70                  self, name, email, password, birthday
71          ):
72              Person.__init__(
73                  self, name, email, password, birthday
74              )
75
76      # Review class
77      class Review:
78
```

```
79          def __init__(
80                  self, idreview, name, value, description
81          ):
82              # "idreview" is an int
83              self.idreview = idreview
84              # "name" is the client's name,
85              # it is a Client type
86              self.name = name
87              # "value" is an int
88              # 1, 2, 3, 4, 5
89              self.value = value
90              # "description" is a string
91              # in which there is the feedback
92              self.description = description
93
94          # getidreview() retrieves the id of the review,
95          # setidreview() changes the id of the review
96          def getidreview(self):
97              return self.idreview
98
99          def setidreview(self, a):
100                 self.idreview = a
101                 return self.idreview
102
103             # getclient() retrieves the client,
104             # setclient() changes the client
105             def getclient(self):
106                 return self.name
107
108             def setclient(self, a):
109                 self.name = a
```

```
110                 return self.name
111
112             # getvalue() retrieves the value,
113             # setvalue() changes the value
114             def getvalue(self):
115                 return self.value
116
117             def setvalue(self, a):
118                 self.value = a
119                 return self.value
120
121             # getdescription() retrieves the text,
122             # setdescription() changes the text
123             def getdescription(self):
124                 return self.description
125
126             def setdescription(self, a):
127                 self.description = a
128                 return self.description
129
130         # Typology class
131         class Typology:
132
133             def __init__(self):
134                 self = self
135
136         # Holidaymaker class,
137         # it is a subclass of Typology
138         class Holidaymaker(Typology):
139
140             def __init__(self):
```

```
141                 Typology.__init__(self)
142
143         # Overnightstay class,
144         # it is a subclass of Typology
145         class Overnightstay(Typology):
146
147             def __init__(self):
148                 Typology.__init__(self)
149
150         # Serviceinloco class,
151         # it is a subclass of Typology
152         class Serviceinloco(Typology):
153
154             def __init__(self):
155                 Typology.__init__(self)
156
157         # Transport class,
158         # it is a subclass of Typology
159         class Transport(Typology):
160
161             def __init__(self):
162                 Typology.__init__(self)
163
164         # TourismProduct class
165         class TourismProduct:
166
167             def __init__(
168                     self, id, nametp, name, visitornum,
169                     birthday, city, typology, surface,
170                     period, arrive, pagerank, prices,
171                     quality, howtoarrive, numreview
```

```
172             ):
173                 # "id" is an int
174                 self.id = id
175                 # "nametp" is a string
176                 self.nametp = nametp
177                 # "name" is a Client type
178                 self.name = name
179                 # "visitornum" is an int
180                 self.visitornum = visitornum
181                 # "birthday" is a string
182                 self.birthday = birthday
183                 # "city" a City type
184                 self.city = city
185                 # "typology" is a Typology type
186                 self.typology = typology
187                 # "area" is an int
188                 self.surface = surface
189                 # "period" is an int
190                 self.period = period
191                 # "path" is a boolean
192                 self.arrive = arrive
193                 # "pagerank" is an int
194                 self.pagerank = pagerank
195                 # "prices" is an int
196                 self.prices = prices
197                 # "quality" is an int
198                 self.quality = quality
199                 # "howtoarrive" is Howtoarrive type
200                 self.howtoarrive = howtoarrive
201                 # "numreview" is an int
202                 self.numreview = numreview
```

```
203
204             # getid() retrieves the id,
205             # setid() changes the id
206             def getid(self):
207                 return self.id
208
209             def setid(self, a):
210                 self.id = a
211                 return self.id
212
213             # getnametp() retrieves the name of the TP,
214             # setnametp() changes the name of the TP
215             def getnametp(self):
216                 return self.nametp
217
218             def setnametp(self, a):
219                 self.nametp = a
220                 return self.nametp
221
222             # getname() retrieves the manager's name,
223             # setname() changes the manager's name
224             def getname(self):
225                 return self.name
226
227             def setname(self, a):
228                 self.name = a
229                 return self.name
230
231             # getvisitornum() retrieves
232             # the visitors number
233             # setvisitornum() changes the visitors number
```

```
234             def getvisitornum(self):
235                 return self.visitornum
236
237             def setvisitor(self, a):
238                 self.visitornum = a
239                 return self.visitornum
240
241             # getbirthday() retrieves the date
242             # of the TP has been started,
243             # setbirthday() changes the date
244             def getbirthday(self):
245                 return self.birthday
246
247             def setbirthday(self, a):
248                 self.birthday = a
249                 return self.birthday
250
251             # getcity() retrieves the city,
252             # setcity() changes the city
253             def getcity(self):
254                 return self.city
255
256             def setcity(self, a):
257                 self.city = a
258                 return self.city
259
260             # gettypology() retrieves the typology,
261             # settypology() changes the typology
262             def gettypology(self):
263                 return self.typology
264
```

```
265             def settypology(self, a):
266                 self.typology = a
267                 return self.typology
268 
269             # getsurface retrieves the surface area,
270             # setsurface() changes the surface area
271             def getsurface(self):
272                 return self.surface
273 
274             def setsurface(self, a):
275                 self.surface = a
276                 return self.surface
277 
278             # getperiod() retrieves the period,
279             # setperiod() changes the period
280             def getperiod(self):
281                 return self.period
282 
283             def setsurface(self, a):
284                 self.surface = a
285                 return self.surface
286 
287             # getarrive() retrieves the boolean value,
288             # setvalue() changes the value
289             def getarrive(self):
290                 return self.arrive
291 
292             def setarrive(self, a):
293                 self.arrive = a
294                 return self.arrive
295 
```

```
296             # getpagerank() retrieves the pagerank value,
297             # setpagerank() changes the pagerank value
298             def getpagerank(self):
299                 return self.pagerank
300
301             def setpagerank(self, a):
302                 self.pagerank = a
303                 return self.pagerank
304
305             # getprices() retrieves prices,
306             # setprices() changes prices
307             def getprices(self):
308                 return self.prices
309
310             def setprices(self, a):
311                 self.prices = a
312                 return self.prices
313
314             # getquality() is the arithmetic average
315             # of the total value of the review
316             # setquality() changes the value
317             def getquality(self):
318                 return self.quality
319
320             def setquality(self, a):
321                 self.quality = a
322                 return self.quality
323
324             # gethowtoarrive() retrieves the value,
325             # sethowtoarrive() changes the value
326             def gethowtoarrive(self):
```

```
327                 return self.howtoarrive
328
329             def sethowtoarrive(self, a):
330                 self.howtoarrive = a
331                 return self.howtoarrive
332
333             # getreview() retrieves the int
334             # of the number of the reviews,
335             # setreview() changes the int
336             def getreview(self):
337                 return self.numreview
338
339             def setreview(self, a):
340                 self.numreview = a
341                 return self.numreview
342
343
344         # City class
345         class City:
346
347             def __init__(
348                     self, big, namect, cinum, visnum
349             ):
350                 # "dimension" is an int,
351                 # we have three value:
352                 # 50 capital
353                 # 30 chief
354                 # 15 town
355                 # 5 village
356                 self.big = big
357                 # "namect" is a string
```

```
358                 self.namect = namect
359                 # "cinum" is an int
360                 self.cinum = cinum
361                 # "visnum" is a int
362                 self.visnum = visnum
363 
364             # getnamect() retrieves the name city,
365             # setnamect() changes the name city
366             def getnamect(self):
367                 return self.namect
368 
369             def setnamect(self, a):
370                 self.namect = a
371                 return self.namect
372 
373             # getbig() retrieves the value,
374             # setbig() changes the value
375             def getbig(self):
376                 return self.big
377 
378             def setbig(self, a):
379                 self.big = a
380                 return self.big
381 
382             # getcinum() retrieves the number of citizens,
383             # setcinum() changes the number of citizens
384             def getcinum(self):
385                 return self.cinum
386 
387             def setcinum(self, a):
388                 self.cinum = a
```

```
389                 return self.cinum
390
391             # getvisnum() retrieves the number
392             # of the visitors,
393             # setvisnum() changes the number of visitors
394             def getvisnum(self):
395                 return self.visnum
396
397             def setvisnum(self, a):
398                 self.visnum = a
399                 return self.visnum
400
401         # Howtoarrive class
402         class Howtoarrive:
403
404             def __init__(
405                     self, frequency, howtoarrive
406             ):
407                 # "frequency" is an int,
408                 # 3 -> 1 min to 1 hour
409                 # 2 -> 1 hour to 3 per day
410                 # 1 -> over 3 per day
411                 self.frequency = frequency
412                 # howtoarrive is an int,
413                 # Plane = 2
414                 # Train or Boat = 3
415                 self.howtoarrive = howtoarrive
416
417
418             def getfrequency(self):
419                 return self.frequency
```

```
420
421             def setfrequency(self, a):
422                 self.frequency = a
423                 return self.frequency
424
425             # gethowtoarive() retrieves the int,
426             # sethowtoarrive() changes the value
427             def gethowtoarrive(self):
428                 return self.howtoarrive
429
430             def sethowtoarrive(self, a):
431                 self.howtoarrive = a
432                 return self.howtoarrive
433
434
435
436         # Appeal returns an int
437         # Appeal is the value of the Tourism Product
438         def Appeal(tourism):
439
440             # Parameters used in the weighted average
441             # CITY
442             ct = tourism.getcity().getbig()
443             # CAPACITY
444             su = tourism.getsurface()
445             vs = tourism.getvisitornum()
446             capacity = su/vs
447             # EVALUATION
448             ev = tourism.getreview()
449             qu = tourism.getquality()
450             evaluation = ev/qu
```

```
451             # PAGERANK
452             pa = tourism.getpagerank()
453             # WAY
454             wy = tourism.gethowtoarrive()
455             way = 0
456             # Check if the Tourism Product
457             # has the parameter
458             if wy == False:
459                 way
460             elif wy == True:
461                 # WAY
462                 w1 =
463                     tourism.gethowtoarrive()
464                     .getfrequency()
465                 w2 =
466                     tourism.gethowtoarrive()
467                     .gethowtoarrive()
468                 way = w1 + w2
469             # PRICES
470             pr = tourism.getprices()
471             # PERIOD
472             pe = tourism.getperiod()
473             # We differentiate the different typologies
474             # because the prices change
475             if tourism.gettypology() == Serviceinloco:
476                 prices = pr / pe
477             elif tourism.gettypology() == Holidaymaker:
478                 prices = (su / pr) / pe
479             elif tourism.gettypology() == Overnightstay:
480                 prices = pr / pe
481             else:
```

```
482                 prices = pr / (tourism.getvisitornum() +
483                               tourism.getcity().getcinum())
484
485             # We take seven parameters
486             # to have the final Appeal
487             # "ct" capital/chief/town/village
488             # "pe" is the opening months
489             # "prices" is the arithmetic average
490             # of the prices
491             # "capacity" is the relationship between
492             # "surface area" and "number of visitors"
493             # "evaluation" is the relationship between
494             # "number of evaluation" and
495             # "average of the quality"
496             # "way" is the result
497             # of "frequency" and "transport"
498             # "pa" is the result of PageRank
499             appeal =
500                     (ct + pe + prices +
501                     capacity + evaluation +
502                     way + pa) / 7
503
504             print("The final Appeal of " +
505                     tourism.getnametp() +" is:")
506             print(appeal)
507             print('\n')
508
509         # We compare two Tourism Products
510         # with the same Typology
511         def Compare(t1, t2):
512
```

```
513             Appeal(t1)
514             Appeal(t2)
515
516             # Parameters of the first TP
517             # CITY t1
518             ct1 = t1.getcity().getbig()
519             # PERIOD t1
520             pe1 = t1.getperiod()
521             # CAPACITY t1
522             su1 = t1.getsurface()
523             vs1 = t1.getvisitornum()
524             capacity1 = su1/vs1
525             # PRICES t1
526             pr1 = t1.getprices()
527             # We differentiate the different typologies,
528             # first Tourism Product
529             if t1.gettypology() == Serviceinloco:
530                 prices1 = pr1 / pe1
531             elif t1.gettypology() == Holidaymaker:
532                 prices1 = (su1 / pr1) / pe1
533             elif t1.gettypology() == Overnightstay:
534                 prices1 = pr1 / pe1
535             else:
536                 prices1 = pr1 / (
537                     t1.getvisitornum() +
538                     t1.getcity().getcinum()
539                     )
540             # EVALUATION t1
541             ev1 = t1.getreview()
542             qu1 = t1.getquality()
543             evaluation1 = ev1/qu1
```

```
544             # WAY t1
545             wy1 = t1.gethowtoarrive()
546             # Check if the first Tourism Product
547             # has the parameter
548             way1 = 0
549             if wy1 == False:
550                 way1
551             elif wy1 == True:
552                 w1t1 = t1.gethowtoarrive()
553                         .getfrequency()
554                 w2t1 = t1.gethowtoarrive()
555                         .gethowtoarrive()
556                 way1 = w1t1 + w2t1
557             # PAGERANK t1
558             pa1 = t1.getpagerank()
559
560             # Parameters of the second TP
561             # CITY t2
562             ct2 = t2.getcity().getbig()
563             # PERIOD t2
564             pe2 = t2.getperiod()
565             # CAPACITY t1
566             su2 = t2.getsurface()
567             vs2 = t2.getvisitornum()
568             capacity2 = su2/vs2
569             # PRICES t2
570             pr2 = t2.getprices()
571             # We differentiate the different typologies,
572             # second Tourism Product
573             if t2.gettypology() == Serviceinloco:
574                 prices2 = pr2 / pe2
```

```
575             elif t2.gettypology() == Holidaymaker:
576                 prices2 = (su2 / pr2) / pe2
577             elif t2.gettypology() == Overnightstay:
578                 prices2 = pr2 / pe2
579             else:
580                 prices2 = pr2 / (
581                     t2.getvisitornum() +
582                     t2.getcity().getcinum()
583                     )
584             # EVALUATION t2
585             ev2 = t2.getreview()
586             qu2 = t2.getquality()
587             evaluation2 = ev2/qu2
588             # WAY t2
589             wy2 = t2.gethowtoarrive()
590             # Check if the second Tourism Product
591             # has the parameter
592             way2 = 0
593             if wy2 == False:
594                 way2
595             elif wy2 == True:
596                 w1t2 = t2.gethowtoarrive()
597                         .getfrequency()
598                 w2t2 = t2.gethowtoarrive()
599                         .gethowtoarrive()
600                 way2 = w1t2 + w2t2
601             # PAGERANK t2
602             pa2 = t2.getpagerank()
603
604             # We compare the parameters of both TPs
605             while True:
```

```
606
607                 # First control,
608                 # if the TPs are different,
609                 # we can not compare them
610                 if t1.gettypology() != t2.gettypology():
611                     print("We can't compare them!")
612                     print('\n')
613                     break
614
615                 # We compare the city
616                 # where the TPs are located in
617                 if ct1 > ct2:
618                     print("CITY")
619                     print(
620                         "The City "
621                         + t1.getnametp()
622                         + " is bigger"
623                         )
624                     print('\n')
625                 elif ct2 > ct1:
626                     print("CITY")
627                     print(
628                         "The City "
629                         + t2.getnametp() +
630                         " is bigger"
631                         )
632                     print('\n')
633                 elif ct1 == ct2:
634                     print("CITY")
635                     print(
636                           "The cities are big both"
```

```
637                             )
638                     print('\n')
639
640                 # We compare the period of opening
641                 if pe1 > pe2:
642                     print("PERIOD")
643                     print(
644                         "The opening months of "
645                         + t1.getnametp() +
646                         " are bigger then "
647                         + t2.getnametp()
648                         )
649                     print('\n')
650                 elif pe2 > pe1:
651                     print("PERIOD")
652                     print(
653                         "The opening months of "
654                         + t2.getnametp() +
655                         " are bigger then "
656                         + t1.getnametp()
657                         )
658                     print('\n')
659                 else:
660                     print("PERIOD")
661                     print(
662                           "The opening months "
663                           "of the TPs are equal"
664                           )
665                     print('\n')
666
667                 # We compare the arithmetic averages
```

```
668                 # of the prices
669                 if prices1 > prices2:
670                     print("PRICES")
671                     print(
672                         t1.getnametp() +
673                         " is more expensive than "
674                         + t2.getnametp()
675                         )
676                     print('\n')
677                 elif prices2 > prices1:
678                     print("PRICES")
679                     print(
680                         t2.getnametp() +
681                         " is more expensive than "
682                         + t1.getnametp()
683                         )
684                     print('\n')
685                 else:
686                     print("PRICES")
687                     print(
688                           "The prices are equal"
689                           )
690                     print('\n')
691 
692                 # We compare the capacities
693                 if capacity1 > capacity2:
694                     print("CAPACITY")
695                     print(
696                           "The capacity of "
697                           + t1.getnametp() +
698                           " is bigger than "
```

```
699                         + t2.getnametp()
700                     )
701                 print('\n')
702             elif capacity2 > capacity1:
703                 print("CAPACITY")
704                 print(
705                     "The capacity of "
706                     + t2.getnametp() +
707                     " is bigger than " +
708                     t1.getnametp()
709                     )
710                 print('\n')
711             else:
712                 print("CAPACITY")
713                 print(
714                       "The capacities are equal"
715                       )
716                 print('\n')
717
718             # We compare the evaluations
719             if evaluation1 > evaluation2:
720                 print("EVALUATION")
721                 print(
722                       "The evaluation of "
723                       + t1.getnametp() +
724                       " is bigger than " +
725                       t2.getnametp()
726                     )
727                 print('\n')
728             elif evaluation2 > evaluation1:
729                 print("EVALUATION")
```

```
730                     print(
731                          "The evaluation of " +
732                           t2.getnametp() +
733                           " is bigger than " +
734                           t1.getnametp()
735                         )
736                     print('\n')
737                 else:
738                     print("EVALUATION")
739                     print(
740                         "The number evaluations are equal"
741                         )
742                     print('\n')
743
744                 # We compare way parameter
745                 if way1 > way2:
746                     print("WAY")
747                     print(
748                         t1.getnametp() +
749                         " is easier to reach"
750                         )
751                     print('\n')
752                 elif way2 > way1:
753                     print("WAY")
754                     print(
755                         t2.getnametp() +
756                         " is easier to reach"
757                         )
758                     print('\n')
759                 else:
760                     print("WAY")
```

```
761                     print(
762                         "The TPs have"
763                         "the same way to reach"
764                         )
765                     print('\n')
766 
767                 # We compare the PageRank results
768                 if pa1 > pa2:
769                     print("PAGERANK")
770                     print(
771                         "The PageRank result is"
772                         "bigger in " +
773                         t1.getnametp()
774                         )
775                     print('\n')
776                 elif pa2 > pa1:
777                     print("PAGERANK")
778                     print(
779                         "The PageRank result is"
780                         "bigger in " +
781                         t2.getnametp()
782                         )
783                     print('\n')
784                 else:
785                     print("PAGERANK")
786                     print("The PageRank results are"
787                           "the same")
788                     print('\n')
789 
790                 break
791 
```

```
792         # Some examples
793
794         # Managers
795         ben = Manager(
796             "Ben", "ben@ulli.com", "benebnene", "7/8/1978"
797         )
798         cam = Manager(
799             "Cam", "cam@cam.com", "blalalalbal", "5/9/1956"
800         )
801
802         # Clients
803         trilli = \
804             Client(
805                 "Trilli", "trilli.miao@trilli.com",
806                 "miaomiao", "17/5/2005"
807             )
808         giu = \
809             Client(
810                 "Giulia", "giugiu@",
811                 "gattomiao", "16/02/1989"
812             )
813
814         # Reviews
815         a = \
816             Review(
817                 23, trilli, 5, "this is fantastic"
818             )
819         b = \
820             Review(
821                 45, giu, 1, "not good"
822             )
```

```
823
824         # Cities
825         london = \
826             City(
827                 50, "London", 789564215458, 78787881545131
828             )
829         venice = \
830             City(
831                 50, "Venice", 78844154478, 5887818989898
832             )
833         smallville = \
834             City(
835                 30, "Smallville", 89890, 909980
836             )
837         cicut = \
838             City(
839                 15, "Cicut", 8989, 76767
840             )
841
842         # How To Arrive
843         ha1 = \
844             Howtoarrive(
845                 3, 2
846             )
847         ha2 = \
848             Howtoarrive(
849                 2, 4
850             )
851         ha3 = None
852
853         # Tourism Products
```

```
854         jetmarket = \
855             TourismProduct(
856                 78, "JetMarket", ben, 78778,
857                 "8/8/1988", smallville, Serviceinloco,
858                 56547, 12, True, 5, 7878, 5, ha1, 89
859             )
860         sunmall = \
861             TourismProduct(
862                 5, "SunMall", cam, 7455521,
863                 "8/5/2012", venice, Holidaymaker,
864                 78878, 12, True, 26, 8568774, 5, ha1, 85
865             )
866         fun = \
867             TourismProduct(8, "FUn!", ben, 855151,
868                            "5/5/2005", london,
869                             Serviceinloco,
870                             555, 4, False,
871                             89, 74, 3, ha3, 25
872                            )
873
874         #Appeal(fun)
875         #Compare(sunmall, jetmarket)
876         Compare(jetmarket, fun)
877
878     if __name__ == "__main__":
879     main()
```

Code 6 ~ Database