

SCHOOL OF COMPUTER SCIENCE
THE UNIVERSITY OF MANCHESTER

Picalag : an intelligent calendar aggregator

A dissertation submitted to the University of Manchester for the degree of
Master of Science in the School of Computer Science

2011

Sébastien GRILLOT

MSc Advanced Computer Science and IT Management

Contents

1	Introduction	10
1.1	Project overview and motivations	10
1.2	About the following document	12
2	Background	13
2.1	Calagator	13
2.1.1	Presentation	13
2.1.2	Aggregation: import and export supported formats	14
2.2	Recommender systems (RSs)	16
2.2.1	Definition	16
2.2.2	Different types of Recommender Systems	18
2.2.3	Criteria for the choice of relevant RS	27
2.3	Related works: Events recommendation	32
3	Design and Implementation	35
3.1	Introduction	35
3.2	Requirements Analysis	36
3.3	Project organisation	38
3.3.1	Deliverables	38
3.3.2	Research and development methodology	39
3.4	Tools	41
3.5	Architecture	45

3.5.1	The automated events scraping and import scripts	46
3.5.2	The front end web-application	47
3.5.3	The PServer back end application	54
3.6	Deliverables	68
4	Evaluation and analysis	69
4.1	Introduction	69
4.2	Evaluation methodology	69
4.3	Evaluations	71
4.3.1	Evaluation 1: relevance of the intelligent part	71
4.3.2	Evaluation 2: data structure, storage and performance	73
4.3.3	Evaluation 3: user feedback	74
4.4	Results	76
4.4.1	Participants sample	76
4.4.2	Data feed	77
4.4.3	Data structure, storage and performance	78
4.4.4	Relevance of the intelligent part and user feedbacks	81
5	Conclusion	94
5.1	Summary	94
5.2	Future work	96
	Bibliography	100
A	Evaluation questionnaires	107
A.1	Initial form	107
A.2	Picalag evaluation - End of session Form	112
A.3	Picalag Evaluation: Final form	114

B Documentation	118
B.1 Picalag Front-end application	118
B.1.1 Installation instructions	118
B.1.2 Import RESTful API	120
B.2 PServer Back-end application	121
B.2.1 Installation instructions	121
B.2.2 RESTful API	122
Final word count (including appendices):	31,726

List of Figures

3.1	Constructive research approach	40
3.2	Development cycle diagram	40
3.3	Picalag global architecture	46
3.4	Picalag front end architecture	48
3.5	Manage favorite venues list widget - even page (before/after click) . .	52
3.6	Event rating widget, share widget and export function	54
3.7	Three-tiers model	55
3.8	PServer request processing path	57
3.9	PServer web.xml config file	57
3.10	PServer database diagram	58
3.11	Items Manager sub-system	61
3.12	User profile sub-system	63
3.13	Recommender systems sub-system	65
4.1	User profile saturation	73
4.2	Content-based RS response time	79
4.3	Evolution of number of features in PServer and in a user profile . . .	80
4.4	Evolution of PServer Database size	80
4.5	Recommendation modules ranking	83
4.6	Relevance of recommendations	84
4.7	Match recommendations - user interests	85
4.8	Coherence of the recommendations	85

4.9	Evolution of the quality of the recommendations	86
4.10	Recommendations grade	86
4.11	Is the interface comfortable?	87
4.12	Trust in the system	88
4.13	Control over the system	88
4.14	Quality of the interface	88
4.15	Responsiveness of the system	89
4.16	Usefulness of the system	89
4.17	Users opinions	90
4.18	System Usability Scale	92
4.19	SUS score interpretation [70]	93

List of Tables

2.1	Domain factors and recommendation techniques [10]	30
2.2	Picalag specification analysis for the choice of the relevant system	31
4.1	Events importation	77
4.2	Recommendations during the experiment	83

Abstract

With the development of Internet, we entered the so called information age: information is everywhere and cannot be filtered by humans any more. To address this “information overload” problem, it is common to be automatically helped in decision making process by recommender systems. However, only few systems exist to help to find social events.

This project, *Picalag* (for *Personal Calendar Aggregator*), is a web-based calendar aggregator for events happening in Manchester area, augmented with a recommender system to assist user in the choice of events to attend. The final aim is the improvement of user experience in comparison with other existing services that do not offer recommendations.

Based on the open-source project Calagator, the developed system aggregates events from different calendar data sources to offer the user the most complete database possible. Recommendations are then made based on previously browsed and liked events: the system uses their contents to find similar items in the database (content-based filtering). Items are also recommended if like-minded users showed interest for them (collaborative filtering).

Implemented in ten weeks, the two main independent parts of the system are designed to be cross-platform and scalable: a Ruby on Rails web-interface is completed by a dedicated personalisation server (PServer) powered by Java. Evaluated by 14 users, it proved to be functional and easy to use.

Keywords: Picalag, Calendar aggregator, Recommender system, Events recommendation

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Intellectual property statement

The author of this dissertation (including any appendices and/or schedules to this dissertation) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

Copies of this dissertation, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has entered into. This page must form part of any such copies made.

The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the dissertation, for example graphs and tables (“Reproductions”), which may be described in this dissertation, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

Further information on the conditions under which disclosure, publication and commercialisation of this dissertation, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see <http://documents.manchester.ac.uk/display.aspx?DocID=487>), in any relevant Dissertation restriction declarations deposited in the University Library, The University Library’s regulations (see <http://www.manchester.ac.uk/library/aboutus/regulations>) and in The University’s Guidance for the Presentation of Dissertations.

“Everybody gets so much information all day long that they lose their common sense.”

Gertrude STEIN

“Information is a source of learning. But unless it is organized, processed, and available to the right people in a format for decision making, it is a burden, not a benefit.”

William POLLARD

Acknowledgement

First of all I would like to thank my parents, without whom none of this would have been possible.

I would also like to thank everybody who supported me during this MSc Degree and who shared time with me to make this year interesting and fulfilling, as well as all the participants who helped me to evaluate the system during the experiment.

And obviously I would like to thank my supervisor Prof. Alasdair Rawsthorne for his friendly support, advices and enthusiasm all along this project.

Sébastien GRILLOT - September 2011

Chapter 1

Introduction

1.1 Project overview and motivations

Anybody having various interests and a fulfilling social life in this city has noticed that it is sometimes frustrating to find upcoming events in Manchester. This frustration feeling might be caused by the spreading of information over various web sources. Indeed, users often have to browse numerous event feeds to find the upcoming event that will suit them best (e.g. Citylife[15], Upcoming[75]). This observation is even worse when we consider research conferences, talks or seminars as (nearly) each academic department has its own unformatted list of events (not always up to date).

The purpose of this MSc dissertation calendar aggregation project, Picalag, is to solve this concrete problem by developing a unique website to aggregate several major event feeds in Manchester and surroundings. Such aggregator websites already exist for other cities. For instance, the technology community of Portland, Oregon, developed an open source solution called Calagator [12] that will be used as a base for this project.

Nevertheless, by aggregating different sources, a new problem known as “information overload” will be likely to appear. This is also a frustrating issue as too much information is harmful to decision making. As we do not just want to replace a disappointment with an annoyance, the out coming solution will have to help the final user in his choices thanks to a recommendation system based on his tastes and those of like-minded users.

Recommender systems are still an active research field in computer science. They

proved to be very valuable in commercial applications such as the famous Amazon which contributed to make them famous. Other classic systems use them to reduce the information overload issue: many news readers and RSS aggregators are widely and successfully relying on this technology. However, there is no unique answer as far as recommendations are concerned. Each problem is different and requires to be treated with all its specificities. For this reason, a recommender system used for news articles could not be re-used as is to generate events recommendations. Many different techniques exist, more or less adapted depending on the nature of items to recommend and the data available as input. In the majority of cases, content-based systems (that use the content to recommend items) and collaborative filtering systems (which base their recommendations on users ratings and actions) are used or mixed together in hybrid systems. It is then important to spend some time understanding the strengths and weaknesses of each possible algorithm before selecting the one(s) adapted to this project.

Even if recommender systems are common, used everywhere and with nearly every kind of items over the web, only few implementations operate on events.

Our research hypothesis is that user's experience in choosing social events to attend can be improved by an aggregation system that is able to give recommendations based on his personal preferences. To that end, an adaptive and personalised search and browsing interface layer will be added to Calagator. To integrate it, the interface elements that will be developed will have to be designed keeping Calagator philosophy in mind: they should be minimalist, effective and user-friendly. Moreover, a good recommendation system is a system that you hardly notice as it should be unobtrusive: user does not like to mark or comment (because it is time consuming). The system should then be able to collect implicit feedbacks of the user to build his profile.

This is a really challenging part of this project as calendar information are data with a relatively short life time (when the event is passed, the information is no more relevant). Furthermore, little ranking data is available as the users often cannot comment or mark it objectively before they physically attend the event and then, the information is not useful any more so the user will probably not go back to the page and give his opinion.

1.2 About the following document

This Dissertation is composed of 3 chapters which present the MSc project Picalag. In chapter 2, relevant background literature research for this project is reviewed. Different techniques of recommendation are detailed and preliminary study is carried out to choose and adapt the proposed techniques to the particular problem of events recommendation.

Chapter 3 deals with the actual design and implementation of the system. It describes the choices made when building the system as well as each component developed to make it come to life.

Finally, chapter 4 is devoted to the evaluation of the system. The experiment is described, with hypotheses, objectives and questions it solves. Results are then presented, analysed and discussed to highlight possible future work and ways of improvement for the system.

Chapter 2

Background

This chapter is devoted to background researches and review of relevant literature for this project. It is organised in three sections: the first one (2.1) presents Calagator web-application that will be the base for Picalag project. The second one (2.2) explains how recommender systems work and provides a survey of various literature sources in the field. Section 2.3 finally gives an insight into similar projects and works developed so far.

2.1 Calagator

2.1.1 Presentation

Calagator [12] is used as a base for the development of this project. It is an open-source project started in 2008 with one main goal: building a unified calendar web-application of tech events in Portland, Oregon, United-States. The name Calagator obviously comes from *Calendar aggregator*. Indeed, this application offers the user several methods to create, modify, but also import events from various event feeds over the web. It also gives the opportunity to export events in several standards formats such that iCalendar or hCalendar. These formats are described in next subsection.

The application is developed with Ruby on Rails framework, powered by Ruby programming language which is known to be an efficient and powerful solution for web development (see section 3.4 on page 41 for more details about this framework).

The community is fairly active according to their Google Code project page[28] and updates are still regular thanks to about twenty committers.

It uses several APIs and acts as a mash-up application to output as much useful information about calendar as possible. For instance, Calagator features GoogleMaps link and integration to print venue spot on a map, with then the possibility to access Google maps specific tools, e.g. to compute access itinerary. A brand new stable version also includes Plancast RSVPs [60] (from French “ *Répondez s’il vous plaît*”, Reply please) that allow users to confirm their attendance thanks to their favorite social network account and display their picture on the page of the event.

2.1.2 Aggregation: import and export supported formats

Several common formats and standards are supported by Calagator. It is also built to be able to import feeds from several famous events websites as it has been specified to be an aggregator. This subsection gives details about these formats.

iCalendar

iCalendar is a standard file format for calendar data exchange created by the Internet Engineering Task Force (IETF) Calendaring and Scheduling Working Group and specified in RFC5545[34] (September 2009). The current version is based, for compatibility reasons, on vCalendar format proposed by the Internet Mail Consortium. An iCalendar file is plain text and often presents the *.ics filename extension. It was designed to share easily calendar information and events (e.g. meeting requests) between internet users and is widely supported by all major calendar applications (Google calendar, Yahoo! calendar, Apple iCal, Microsoft Outlook, IBM Lotus notes...). This support is part of its adoption and success.

Basic file structure is composed of lines with the format “FIELD:value”, and the calendaring and scheduling information strictly speaking (a.k.a. iCalBody) is wrapped between “BEGIN:VCALENDAR” and “END:VCALENDAR” tags.

Sometimes, iCalendar is extended with vCard fields as it is a mean to express some information such as venue contacts (e.g. phone number, mail, address) more precisely.

hCalendar

HCalendar 1.0 stands for “HTML iCalendar” and is an open-standard micro-format [49] commonly used to embed iCalendar information in an xHTML web-page so that it is both human and machine readable. To achieve this goal, invisible `` tags and class attributes are added in the HTML markup. The process is completely transparent for human user as the content is displayed as if these tags were not added. On a machine point of view, however, the event can now be easily parsed and extracted and several libraries exist in various programming languages.

This format is used by several leading internet actors (among others Google, Upcoming, Facebook and Wikipedia). Thus, supporting hCalendar format is important for interoperability and aggregation purposes.

Upcoming

The website Upcoming [75] is powered by Yahoo!. It is a largely used event guide database with a developer API that allows developers to search and parse events data for mash-ups or aggregation purposes. As mentioned above, this website also supports hCalendar micro-format. However, the developers API offers more data than a simple hCalendar markup. That is why a specific parser was built in Calagator: when an Upcoming link is imported, the application extracts the event ID and uses the API to get an XML instead of simply parsing the hCalendar information on the HTML page.

The API is RESTful [76], that means that methods are mapped behind simple URL routes and can be called with basic HTML requests (e.g. GET and POST). Attributes for these methods such as the API Key are sent via the URL with classic GET model. The server then replies with either an XML or a JSON file than can be parsed to get data it wraps back. The XML schema for the reply is not a standard one and was specially designed by Yahoo! for Upcoming use. However, libraries are provided in several programming languages (including Ruby).

Output formats

Output formats are roughly the same as input formats as these are the main standards for calendaring information sharing: iCalendar *.ics file, hCalendar markup.

It can also directly be exported as Google Calendar information thanks to a direct link to Google website with all information included as GET attributes.

Two feeds are also possible: iCalendar and Atom feeds. These allow to keep other potential applications up to date thanks to feed readers. ICalendar feed was sketched above, and Atom was proposed as an alternative standard to RSS2.0 by IETF and specified in December 2005 in RFC4287[33]. Atom feeds are based on XML format and are supported by many important web feed readers, even if RSS is still more spread on the Web.

2.2 Recommender systems (RSs)

Picalag project main goal is to develop a recommender system for events. The literature background research presented in this section explains what is such a system. First, in 2.2.1, the general notion is defined, as well as its history and famous applications. Then, the following subsections, 2.2.2 to 2.2.3, contain a more detailed survey about recommender systems (also know as RSs): the different techniques used to implement them, technical facts about these systems and how to choose the relevant technique for a specific application domain.

2.2.1 Definition

In the Encyclopedia of Machine Learning, it is simply stated that “the goal of a Recommender System is to generate meaningful recommendations to a collection of users for items or products that might interest them” [48]. Indeed, user has often not sufficient knowledge of items available in an information system to meet his or her needs. In daily life, we often rely on other people to recommend products or activities. With information technology, this natural social process was translated in machine learning applications, called recommender systems [62].

These systems aim to help the user to face “information overload”. This term describes the difficulties human has in front of a large amount of information [74]. Indeed, as the quantity of contents grows exponentially on the web and as websites (especially aggregators that are quiet popular) tend to display more and more information, a human user is no more able to filter this data alone. In short, these systems try to give the user suitable information at the suitable moment, seamlessly and in a

personalised way. By some aspects, recommender systems can also be considered as personal assistant to help users in their choice.

In [72], Barry Schwartz discussed the paradox of choice: it is a positive thing to have choice but it implies freedom and relies on user's autonomy and determination such that, when all these responsibilities become excessive, user struggles and have a negative feeling about it (it then clearly becomes counter-productive). As a consequence, commercial applications for these systems are obviously close to marketing issue: when Amazon displays items that "other people liked" their goal is to increase sales and user satisfaction. But they also can be used in non-commercial ways by recommending for instance cultural goods (music, films, books) the way IMDB does or news to read. In any cases, the recommender system is implemented to help the user in decision-making process by linking the available information to his interests [72].

If the system is successful, the experience of the user is improved. User is then more likely to come back and spend more time on the system (this can have substantial impact on advertisement-based revenues). On business side, it also mean that the system can collect more personal data about the user, such as their preferences, demographic details and so on. These can be sufficient reasons why one would want to implement and maintain such a system despite it could be costly [62, 23]. Therefore, recommender systems evolved to meet both user and company needs [48].

Development of RSs is a recent field of research in comparison with other well know fields in computer sciences. The first commercial recommender system was implemented by David Goldberg and his team in 1992 as an innovative mail system called Tapestry [26]. The real application of such systems rocketed from the mid-90's as giant distributed applications emerged over the World Wide Web. We can consider that the internet boom and bust in 2000 was a major event in the development of these technologies. Nowadays, these applications are widely and seamlessly used to provide relevant and personalised content to a user. Indeed, these systems have become increasingly popular during the last decade, pushed by commercial usage were they proved they can be highly valuable. Recommender systems can be found everywhere across the web. The most famous examples are Amazon or Google suggest, as their scale is impressive. In 2006, a movie rental service, Netflix, started a contest with a one million dollars prize to encourage researchers to find ways to improve the accuracy of its predictions [54] (the prize was awarded on September 2009 to team "*BellKor's Pragmatic Chaos*").

To sum up, RSs are designed to predict user interest (or rating) for an item, given previous ratings for other items of him/her or other users in order to push likeable items to him/her as recommendations.

The problem was formally written in [4]. If \mathcal{U} is the set of all users and \mathcal{I} the set of items that can be recommended, there exists a function $f : \mathcal{U} \times \mathcal{I} \rightarrow \mathcal{R}$ where \mathcal{R} is a totally ordered set (e.g. 5 grades scale, binary values) that measures the utility $f(u, i)$ of item $i \in \mathcal{I}$ for user $u \in \mathcal{U}$. In practice, values for this function are not known unless the user rated the item (explicitly or implicitly), and f is only defined over a small subset of $\mathcal{U} \times \mathcal{I}$ (we also can say that the *user-item rating matrix* $[U]$ build with items as column, users as lines and ratings as coefficients is sparse [48]). A recommender system is a program that gives approximated item solutions $s_i \in \mathcal{I}$ for the maximum the function f for all users by computing an extrapolation for the function f . The perfect system would return the item $s'_u \in \mathcal{I}$ personalised for user u such that:

$$\forall u \in \mathcal{U}, \quad s'_u = \arg \max_{i \in \mathcal{I}} f(u, i)$$

2.2.2 Different types of Recommender Systems

According to the definition given above, “recommender systems” is a generic term to speak about applications that provide user recommendation about items of interest. So far, no details were given about how this goal can be achieved: it is the aim of this section.

In fact, there exists a wide range of recommendation techniques. [51, 23] give a complete taxonomy of recommender techniques commonly used. Seven different types can be identified: non-personalised, content-based (CB), collaborative filtering (CF), demographic, knowledge-based, community-based and hybrid systems. Most popular techniques are content-based, collaborative filtering and hybrid systems as they are well known and efficient, even if they still are under active research [48]. Only useful techniques for this project will be explained in the following paragraphs.

Non-personalised RSs

This type of recommender system is not very useful for this project so it is mentioned here just as a reminder. As their name implies, these RSs does not require user profiling and modelling. Thus, they are often very easy to implement. Famous examples

for such RSs is top-10 list (e.g. most visited pages), list of new added items, or even static advertisement to recommend a particular product. We can consider them as early implementations of advanced RSs. The idea behind these systems is: if many people liked this item, it is likely that the new user will also be interested in them.

Content-based filtering (CB)

A content-based recommender system bases its results on items the *active user* (i.e. the user who is going to be recommended items) liked in the past. By doing so, it can recommend items close to user taste: relevant items. For instance, if a user liked both *E.T.: The Extra-Terrestrial* and *Jurassic Park*, a content-based filtering system can infer that this user is a fan of STEVEN SPIELBERG and can recommend *Catch Me If You Can*. It also can infer that the user likes sci-fi movies and recommend *Inception*.

This technique is often compared to information retrieval and information filtering as these fields of research were undoubtedly its roots: user is looking for information [57]. In this case, the user preferences are regarded as a query and items are scored according to their relevance to this query [48] exactly the same way Google ranks pages when one type a query on the search input text box. Moreover, CB systems are often used to recommend items with textual information such as articles or web pages as content can quickly become costly to extract from other kind of data (e.g. music or images): that makes the technique clearly close to information retrieval issue.

For this reason, it is common to use the Vector Space Model (VSM), introduced in [66] as part of information retrieval (IR) research, to represent items and user preferences. Items have features (e.g. for a movie it could be fields as title, director, synopsis) that are used to calculate similarity between them [23]. These features can be internally represented as a vector. This vectorial representation can then be used to compute a similarity distance between two distinct items thanks to neighbourhood-based techniques [20]. A typical distance when using vector space model is the cosine similarity [68].

This means we consider that each item $i \in \mathcal{I}$ can be described as a vector \mathbf{x}_i , and user's preferences can be described as \mathbf{x}_u based on the vectors of items $i \in \mathcal{I}_u$ (with $\mathcal{I}_u \subset \mathcal{I}$, set of items rated by user $u \in \mathcal{U}$) rated by this user and the grades $r_{u,i}$ they

got. One possible formula is

$$\mathbf{x}_u = \sum_{i \in \mathcal{I}_u} r_{u,i} \mathbf{x}_i.$$

This vector \mathbf{x}_u can then be compared using cosine distance to any vectors \mathbf{x}_i , with $i \in \mathcal{I} \setminus \mathcal{I}_u$ to calculate the distance between object i and user's tastes. Items that have the highest similarity can then be recommended to the user. With the cosine similarity introduced above:

$$\text{sim}(\mathbf{x}_u, \mathbf{x}_i) = \cos(\mathbf{x}_u, \mathbf{x}_i) = \frac{\mathbf{x}_u \cdot \mathbf{x}_i}{\|\mathbf{x}_u\| \|\mathbf{x}_i\|} \quad (2.1)$$

For text contents, it is usual to represent features as weights over the set of keywords (a.k.a. bag of words). Weights are classically computed thanks to the *tf.idf* method, also presented in [68]. *Tf.idf* stands for “Term frequency-Inverse document frequency” and is a weighting technique used to determine relevant keywords in a document. It is based on simple observations [57]: a relevant term for a document (i.e. it is linked to the topic of the document) is likely to be a term that appears often in the text of this document (*tf*) but rarely in the whole corpus (*idf*). There also is the idea of normalization on this technique such that long documents are not considered as more or less relevant as the short ones. For instance, the term “the” will appear very frequently in any document, but has to be disregarded because it is not a relevant keyword: if one only base its weight on a document, “the” will be considered important, whether if this is compared with the whole corpus one can say that this word is too common to be linked to the topic of a particular document.

This can be written in term of formulas as proposed in [57]:

To calculate the weight of the term t_k in the document d_j , the TF-IDF function can be used:

$$\text{TF-IDF}(t_k, d_j) = \text{TF}(t_k, d_j) \times \text{IDF}(t_k)$$

With:

$$\text{TF}(t_k, d_j) = \frac{n_{k,j}}{\max_z n_{z,j}} \quad \text{or} \quad \text{TF}(t_k, d_j) = \frac{n_{k,j}}{\sum_z n_{z,j}}$$

Indeed, several ways to calculate TF can be found in the literature, only the two most common are reported here. In both case $n_{k,j}$ is the frequency (i.e. the number

of occurrences) of the term t_k in the document d_j and z index covers all terms t_z in d_j . And:

$$\text{IDF}(t_k) = \log \frac{|D|}{n_k}$$

Where $|D|$ is the total number of documents in the corpus and $n_k = |\{d \in D : t_k \in d\}|$ is the total number of documents that contains the term t_k (i.e. such that $n_{k,j} \neq 0$). We can notice that if the term does not occurs in any document, there is a division by 0. For this reason, $n_k = 1 + |\{d \in D : t_k \in d\}|$ can also be found in some papers. Then weight $w_{k,j}$ is computed with the formula:

$$w_{k,j} = \frac{\text{TF-IDF}(t_k, d_j)}{\sqrt{\sum_{s=1}^{|T|} \text{TF-IDF}(t_s, d_j)^2}}$$

where $|T|$ denotes the total number of keywords t_k in d_j . The denominator is only here to normalize weights so that they are in $[0,1]$ interval and the long documents do not have more chance to be recommended than the shorter ones. In some papers, TF-IDF is directly equals to the weight [67]. Then, similarity between two documents can be computed using (2.1):

$$\text{sim}(d_i, d_j) = \frac{\sum_k w_{k,i} \cdot w_{k,j}}{\sqrt{\sum_k w_{k,i}^2} \cdot \sqrt{\sum_k w_{k,j}^2}}$$

As it was discussed before, both items and user profiles can be internally represented as vectors, that are used in the recommendation process.

This approach has at least three clear advantages [57]:

- It does not rely on other users to recommend items. It means that a new user builds his profile independently to other profiled users. This is important as the system does not require a critical number of users to output accurate recommendations. Thus, the system is directly functional.
- It is transparent and can provide explanations to user for the recommendations: e.g. “the system recommends you *Inception* because you like sci-fi movies”. This point is important as the user can understand how the recommendation was computed: this increases his trust in the application.
- A content-based system does not suffer the first-rater problem. Hence, even new items can be recommended to users providing they are similar to previously liked ones. This is not the case with all other approaches.

Nevertheless, it also proves to have several known drawbacks in its simplest version [4, 57, 51]:

- *Limited content analysis*: in some case the features that can be extracted automatically are not enough to provide accurate recommendations. In music or videos, for instance, it is often costly to extract data. In that case, to input features manually in the system is not always practically possible. Moreover, even if IR techniques can be applied (for textual content) the system is not able to judge the quality of the writing (e.g. a poorly written article that uses the same keywords as an other better one will be equally recommended) or the layout of a web-page. In other words, there is a lack of subjective data.
- It also suffers the *overspecialisation problem* (a.k.a. *serendipity* problem). It means that the system tends to always recommend items with a high computed utility according to user's profile, with little chance to have new items that user could like. The system then can become a bit boring for the user that wants some randomness and novelty to open his mind.
- A third issue is known as *new-user problem*. Indeed, a newly signed-up user will have to rate several items before the system can accurately produce some recommendations. In practice this is an important obstacle as the user often does not want to spend time using the system if it does not work perfectly at first time [14]: user is looking for its immediate benefit and is reluctant to use the system if it takes time for it to learn the user's profile.

Collaborative filtering (CF)

The second most famous approach for RSs is collaborative filtering (CF). Unlike content-based approach that only bases its prediction on active user's rating history and similarities between items, CF also processes other users' ratings. In particular, the system tries to identify like-minded users. As ratings and feedback generated by peers (i.e. users) are used, CF RSs give good results when items are not easy to compare (e.g. because of poor content available for CB systems, variety of different kind of items or because the content does not fully reflect the quality of the item) [71, 48, 20]. Typically, this is the case for cultural events[41, 50].

Moreover, such systems are a simple imitation of what people naturally do in real life: when one hesitates to buy an item or attend an event, it is usual to ask for others' point of view and collect like-minded recommendations before making the final decision. This behaviour is important in any step in life and has been observed

in human nature for thousands of years: we rely on other people recommendation. For items that have a highly social profile such as cultural events, movies and even music, it is even more natural to try to get advices from people having similar tastes. Historically, the first system that describes roots of CF is Grundy, presented in [63]. It uses stereotype information about user to match their interests and provide them the correct information. Then, Tapestry [26] allowed to manually select like-minded people for recommendations. At last, several systems truly provided automated recommendations based on other users' interests (e.g. GroupLens for Usenet news[38] and Ringo for music albums and artists [73]).

Because CF approach is working for very diverse types of items and because it is natural, it is very popular and many RSs (if not most of them) implement this technique. However, all these implementation does not necessarily implement similar algorithm: collaborative filtering methods is a general term that can be sub-divided into two main techniques, *neighbourhood-based* (a.k.a. *memory-based* [7] or *heuristic-based* [4]) on the one hand, *model-based* on the other hand.

The first type of these CF techniques, neighbourhood-based, is probably the most popular as it is also the simplest and gives decent results as far as personalised recommendation are concerned. It is also the closest from natural social process (based on comments previously made) and shares some similarities with the CB approach developed above. Both systems are based on similarity distance computation, hence can use the cosine similarity (2.1) from IR literature [68]. The difference in that case is that the system does not compare vectors of tf.idf weights any more, but vectors of user ratings instead [4].

In detail, neighbourhood-based recommendations can be computed according to two different methods: *user-based* and *item-based* [20, 48].

The former way is the exact mimic of natural decision making process when using other people recommendations: the system uses a weighted mean of ratings from people with similar tastes (i.e. who similarly rated objects) to infer active user's interest in a particular item. The weight represents similarity between the active user and other users using the system. To compute this weight between two users $u, v \in \mathcal{U}$, the cosine similarity (2.1) can be used with two vectors $\mathbf{x}_u, \mathbf{x}_v \in \mathbb{R}^{|\mathcal{I}|}$ containing users' rating for items, i.e. such that $\forall i \in \mathcal{I}, x_{u,i} = \{r_{u,i} \text{ if } i \in \mathcal{I}_u, 0 \text{ otherwise}\}$ (same definition for \mathbf{x}_v). Another popular (and more efficient [7]) measure is the *Pearson*

Correlation defined as [48]:

$$\text{PC}(u, v) = w_{u,v} = \frac{\sum_{i \in \mathcal{I}} (r_{u,i} - \bar{r}_u) \cdot (r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in \mathcal{I}} (r_{u,i} - \bar{r}_u)^2 \cdot \sum_{i \in \mathcal{I}} (r_{v,i} - \bar{r}_v)^2}} \quad (2.2)$$

where \bar{r}_u represents means of ratings made by user u . Based on $w_{u,v}$ weights, the system can determine a list of k neighbours (i.e. users close to the active user). To build this user subset $\mathcal{K} \subset \mathcal{U}$, the algorithm can keep users having the k highest weights, or use threshold filtering, for instance [20]. Predictions for rating of item i by active user u , $\tilde{r}_{u,i}$ can then be computed as follow [48]:

$$\tilde{r}_{u,i} = \bar{r}_u + \frac{\sum_{v \in \mathcal{K}} w_{u,v} \cdot (r_{v,i} - \bar{r}_v)}{\sum_{v \in \mathcal{K}} w_{u,v}} \quad (2.3)$$

This method normalises ratings by introducing user mean rating value: this is important as users rate items differently [20]. One may be strict and rarely give the highest grade whereas somebody else can be more lax while grading items.

However, this method suffers some scalability problems. For this reason, another approach for neighbourhood-based CF method is proposed in [45] and used in Amazon.com RS: the *item-based* (a.k.a. *item-to-item*) collaborative filtering. Rather than trying to determine similar users based on their ratings, the system matches active user's previously rated items to similar items (similarity being items that other users did rate together). This system leads to the famous "People who purchased this product also purchased these items" block of recommendation on Amazon and is described by the author as giving better quality recommendation than the classic user-based CF. Formulas for this approach are similar to (2.2) and (2.3), hence are not detailed in this literature review: the interested reader can find them in [20, 48].

Model-based algorithm usually outperform simpler neighbourhood-based vector similarity approach [7], but is heavier to implement. Roots for this concept is proposed by [6]: CF is considered as a classification problem that allows to use known machine learning algorithms. Rather than using raw rating data, a model is computed and implicit features (a.k.a. Latent Factors) are implied thanks to dimensionality reduction methods (e.g. matrix factorisation techniques, singular value decomposition SVD) [6, 39, 79, 48]. Reducing the problem dimensionality can be essential when data is sparse (few ratings available) and to reduce computational complexity. The model represents user-item interaction and is then used with machine learning (ML) systems trained against previously available data. After this training, ML algorithms

are able to predict unknown item usefulness for the user.

Despite the CF approach is, in some cases, a good alternative to get rid of some of the CB shortcomings such as limited content analysis and serendipity problem, it also suffers some drawbacks [51]:

- It has trouble to handle newly added items as nobody rated them. This is known as the *early-rater* or *new-item* problem: as the system needs information about items from other users, a new item cannot be recommended.
- *Sparsity* of data can also be challenging: if few ratings are available, it becomes hard to find neighbours. This can be the case if the number of item is high while the number of user is limited, or if new data is often fed into the system. Taste overlap then becomes harder to discover.
- Eccentric users (i.e. users with unusual tastes) can have problem to get recommendations as no or few neighbours can be found for them.
- A critical mass of user data must be gathered (i.e. lots of users are required) before the system can be effective. That makes the system difficult to test, for instance. It is the *cold-start problem*. Moreover, new users have to rate a certain number of items before predictions become accurate enough to be used effectively, because ratings are typically used to determine distance between users in neighbourhood-based systems.
- There is also a problem to provide explanation for the recommendations. Indeed, an explanation such that “you were recommended this item because Bob1234 has similar taste and liked it” may not convince the user, especially if Bob1234 is an unknown person. In [31], Herlocker and al. compare the way CF RSs work with black boxes: the lack of transparency for these systems can be a problem for user’s trust. Solutions could be to use trusted peers network built thanks to Facebook friends [37], or built by letting the user rate the “trustworthiness” of others [41].

Demographic

Demographic RS technique uses information about user such as language, country, age, gender. This method is not really used alone as it is very minimalist. However, it can be a base for comparing users when tastes for the items highly depend on these criteria. For instance, it is probably an error to recommend a book written in Chinese if the user does not come from China. Demographic RS are not studied a lot in the literature. [58] presents how this data can be used to improve quality of CB and CF RSs. However, this data can be challenging to obtain as user tends to

be reluctant to disclose personal information: finding ways to encourage user to give correct information is still under research. Moreover, demographic-based recommendations cannot adapt to user's specific tastes, especially if they are changing, as they are a generalisation of preferences for all users of similar demographic profile [51].

Hybrid systems

When analysing the previously discussed shortcomings of each of the main techniques, the idea of hybrid system comes naturally. An hybrid system tries to combine different techniques to balance their respective issues in order to have a robust and reliable final system. Most of the time, the aim of an hybrid system is to mix both content-based and collaborative filtering such that the final RS has their strengths while limiting their drawbacks [51].

Indeed, perfect CB systems can be enhanced using by CF techniques. For instance, it was discussed that CB systems suffers from the *limited content analysis* (i.e. lack of subjective evaluation for items) which is not the case for purely CF approach as the latter only relies on user ratings. Data provided by community of users can thus replace the missing information about items content. Introducing some of the CF paradigms in a CB filtering system can also reduce the *serendipity* problem: if CB RSs have trouble to recommend items that differs from user computed tastes (to bring freshness and open user's mind, or to avoid the user to get bored), it is not a problem for collaborative RSs as they can easily recommend items (i.e. novelties) that peers discovered and liked. Finally, when a user has not rated enough items to build a complete enough profile to be used in a CB system, some ratings can directly be inferred (or simply copied) from similar users' profile to diminish the *new-user problem* implications.

Likewise, a pure CF has shortcomings that can be filled by mixing with CB. First, the *early-rater* issue, when the item is new and that few ratings are available, can be solved by evaluating the product thanks to its content: the system does not require user reviews any more to make a recommendation. Similarly, if a user has few neighbours (i.e. he has original taste in comparison with the majority of other users in the system) CB filtering can be used to generate recommendations based on his (quirky) personal taste as this person cannot rely on others' feedback. This observation can also be part of a solution to the *sparsity* problem. Lastly, a CB system does not need lots of users to be effective. Thus, mixing CB and CF can make the system easier to test and limit the *cold-start* problem: as long as the population (or

the number of quality neighbours) did not reach the critical mass for the CF part of the system to work properly, it can still run using its CB module.

Furthermore, demographic data and non-personalised lists can also be used in a first place as a base to address *new-user* and *new-item* problems as it can generate low accuracy (with low personalisation) recommendations thanks to very general information.

Obviously, some of the above-listed shortcomings still remain and mixing the different approaches only reduces each of their drawbacks without solving them entirely. Nevertheless, studies have shown that CB-CF hybridisation is powerful [58].

Different strategies can be found in the literature to implement hybrid systems. A rather complete survey of existing hybrid systems is proposed in [9]. The choice of the relevant strategy may vary from case to case: for instance, movies recommendation is a different problem from toothbrushes recommendation. However, most of them can be classified according to 4 types [4]:

- CB and CF predictions modules are implemented separately, then returned recommendations from both systems are merged (or not) before printing them on user’s screen
- the system is mostly CB but also uses information gathered on other users’ profiles
- the system is mostly CF but also uses content to evaluate usefulness of items
- the RS is implemented thanks to both CB and CF but forms a coherent block rather than two interconnected distinct modules

2.2.3 Criteria for the choice of relevant RS

As it has been discussed in the previous section, several different types of recommender systems can be used to solve a recommendation and personalisation problem. As often when AI and adaptive algorithms are involved, the choice of the relevant system is problem-specific. In the case of RSs, many criteria have to be considered before the designer or the developer can decide what approach is the best for the domain. Indeed, if pure content-base filtering seems adapted for articles recommendation (with lots of content), it may not be the case for other types of items. Moreover, strengths and shortcomings of each technique presented above are often not enough to make the correct choice.

Unfortunately, even if many taxonomies of RSs can be found in the literature, only few of them consider the choice of relevant RS technique by looking at the domain

data specificity and the knowledge available about it. It is the case of [10] that links classic taxonomy of RSs with the domain (i.e. the set of items) they operate over. According to this paper, the relevant approach has to be chosen depending on domain and system characteristics, listed and defined in the following sub-sections.

Items heterogeneity

A set of item is heterogeneous if they are from many types. For instance, a general purpose commercial website can have to recommend movies, music tracks and vegetables. As the information needed to recommend movie and vegetables are not the same, the system design has to be adapted to this situation. Probably, a less extreme example (provided in [10]) is the comparison between a website that only sells digital cameras (homogeneous items) and an e-commerce website where any electronics can be purchased (heterogeneous).

A recommender system that operates over homogeneous items can base prediction on more precise knowledge about this kind of items (to continue with the example of cameras, it could have special fields for zoom and maximum ISO sensitivity). The system is then easier to maintain, whereas it is much more difficult to determine relevant features for a general purpose recommender system.

Risk

RS designer have to keep in mind that whatever the recommended items are, there is a cost for the user. This cost could be real money (for purchasable goods) or time and cognitive cost (for news articles). This cost should influence the choice of the recommendation approach.

Indeed, the risk is important as it has a direct impact on user's tolerance: if the user is to spend £100+ for an item, it is likely that he will not accept false positive recommendations (recommended items that does not fit his interest) ; another example can be a RS for medical diagnostics which cannot tolerate false negatives as it could be dangerous for the patient if the system ignore an important diagnostic possibility. Poor recommendation tolerance is obviously higher when the cost is low.

Churn

If the churn is high for a domain, it means that items are relevant only for a short period. It is typically the case for news items as new items are added each second and become irrelevant quickly (within a day or a week maximum). An electronics reseller has not this problem as devices are expected to be sold over a long period.

A high churn is synonym with sparse user-item rating matrix as users do not have time to follow the continuous stream of new items. Moreover, even if an item gets several ratings, it often becomes irrelevant by this time.

Interaction style

The collected data is not the same (in term of information, quality, accuracy and volume) if the user gives explicit ratings or feedback or if implicit rating is recorded. For instance, explicit ratings can be given through five points or boolean (like/dislike) scale grading form, whereas implicit rating can be inferred by looking at navigation behaviour on the website. The latter is often noisy and not very accurate but it is easier to get a fair volume of data, as no action is required from the user (hence no effort and no annoyance).

Preference stability

In some domains, user's preferences can change very quickly. For instance for news recommendation, user may suddenly be interested in football during the world cup. Conversely, it can be expected that no brutal switch in user's musical tastes will occur.

This criteria is important for model-based RSs as no model can be inferred if previously collected data is no more relevant. If preferences are stable, however, the system can rely on old data with a good confidence.

Scrutability

Are explanations about the recommendation required? This can be important to increase user's trust and understanding about the system. Typically, when the *risk* is high, user wants to be able to judge if the recommendation is reliable. If the system is able to print "we recommended this car because it is blue and you like this colour

(your previous car is blue), it has 5 seats because you told the system you have two children and it has a huge trunk and diesel engine because we know you often travel with lots of luggage”, the user will probably be more likely to buy the car.

Choice of the most adapted technology for Picalag

A summary of these criteria mapped with the technologies adapted in each case have been copied out from [10] in table 2.1. In next paragraphs, specifications for Picalag are compared to this classification to determine the most suitable strategy to adopt.

Factor		Collaborative	Content-B.	Knowledge-B.
Heterogeneity	Low	✓	✓	✓
	High	+	-	-
Risk	Low	✓	✓	✓
	High	-	-	+
Churn	Low	✓	✓	✓
	High	-	+	+
Interaction	Implicit	+	+	-
	Explicit	✓	✓	+
Stability	Stable	✓	✓	✓
	Unstable	-	-	+
Scrutability	Required	-	-	+
	Not required	✓	✓	✓

Symbols meaning:

- + = very adapted
- ✓ = adapted
- - = not adapted

Table 2.1 – Domain factors and recommendation techniques [10]

Items are relatively heterogeneous at first glance, as Picalag will have to recommend concerts as well as medieval fairs, for instance. In the same time, event data structure is well specified and static, and all items will be events. All in all, heterogeneity is fairly high.

As we are dealing with cultural events, risk can be considered as low. One can argue that the cost may be a problem if event entry fee is expensive or if we consider that user can spend a bad evening by following a faulty recommendation. Nevertheless, this can be limited and the user will be considered competent enough to auto-moderate the returned predictions by detecting false positives easily. False negatives

are not a problem either: even if some possibly suitable items are not recommended, Manchester offers enough events to still find something interesting to do.

Domain churn is very high, as sources aggregated offer around 1500+ events per day. Moreover, events are particularly perishable data: once the event has taken place, the information has no more value. This is emphasised by the fact that it is improbable to get explicit feedback before the event was attended (the user cannot grade it as long as he did not go) and, after the event, it is useless to do so (for perishability reason). Thus, sparsity problem is important and the system has to be able to handle efficiently the load of new-items daily imported.

Mainly because of the reason given in previous paragraph (but also because of specification), the user will interact implicitly with the system: an event is considered interesting when the user opens its page, very interesting when the user exports it to his personal calendar and not interesting if the user discards it thanks to a button on the page (neutral otherwise).

User's preferences can be considered as fairly stable, as it is unlikely his tastes suddenly change. Nonetheless, it can be possible that a big event (tournament, festival) temporarily modify interests. This case is limited and can reasonably be ignored.

Scrutability is not essential in Picalag as the risk is low. However, an explanation engine might be implemented as it is important to increase user's trust and satisfaction, hence the popularity of the system (which has been identified as critical if CF approach is considered for this project). It can also help to counter-balance potentially negative aspects mentioned in risk evaluation.

This characteristic review is summed up on table 2.2 and each factor is associated with the relevant technology to handle the situation. With this analysis, it appears that the most appropriate system for this project is an hybrid system with a content-based predominance. Demographic data could be useful to moderate the cold-start (new user) issue.

Factor	Heterogeneity	Risk	Churn	Interaction	Stability	Scrutability
	Fairly high	Low	High	Implicit	High	Welcome
Tech.	CF	CF, CB	CB	CF, CB	CF, CB	CB

Table 2.2 – Picalag specification analysis for the choice of the relevant system

2.3 Related works: Events recommendation

Recommender systems are traditionally used to operate over a wide variety of different items. They have been famously used, among others, to recommend Usenet news [38], movies (as NetFlix, MovieLens or IMDB system [27]), news articles (the most famous example is GoogleNews [56, 69, 17]), music (e.g. [73], last.fm) or to filter e-mails [26]. Nevertheless, only few research or industrial development were made about event recommendation.

In many respects, cultural events recommendation can be compared to news recommendation. With criteria developed in section 2.2.3, it can be established that, in both case, churn is high and risk is low. That is to say many events are regularly added to the database and the information is perishable but the user can be tolerant to poor recommendations. At this point, an extra difficulty should be considered: perishability in the case of events is materialised with a hard deadline. Indeed, if news article can still be interesting for user even if they are out of date, it is not the case for event information. It limits the number of ratings that can be available for each event, as it was argued in these lines and in [11]. Thus, this temporal information is an important factor. Another essential characteristics for events is the location as the user is not likely to attend an event if it is too far from his home. In Picalag, this is not a problem as we strictly limited the system to events taking place in the city of Manchester only. However, some more (geographically) general recommender systems have to consider this issue. Geolocalisation data is generally obtained either with mobile GPS and GSM signal triangulation [13, 44, 61], or using IP address of the incoming connexion[37]. It is probable that the next generation of events RSs will exploit the new geolocalisation API proposed by the W3C, included in HTML5 [77].

Several different approaches to recommend events can be found in the literature. Most of them are based on hybrid systems, in accordance with the conclusion of the analysis drawn in section 2.2.3. However, implementations are very different one from each other.

In [53], the implemented system follows an hybrid approach with separate sub-systems to recommend event on a Finish festival website. However, results are not merged in the end and let the user know exactly from which recommender module the recommendation comes from. It allows an easier integration with a fair clarity for user. In the rendered web page, the user will find several recommendations categories: “similar events” (classic CB filtering, using Tf.Idf method), “popular events” (non per-

sonalised list based on users count for each event), “coming soon” (to present the very next events taking place), “surprising events” (based on rarely browsed events, to address serendipity problem), and “similar users go” (classic CF). A “random events” section was also created. Statistics for this website showed that users often relied on “popular events” to make their choice as items presented in this category were the most clicked, followed by “similar users go” items. Moreover, users tend to spend more time on the website and browse more pages, as they find interesting information.

[11] is also convinced that hybrid RS is the best solution for events recommendation. Indeed, in this paper, the authors point out the fact that classic collaborative filtering and content-based recommender systems are not really efficient when they are applied to this domain. The former system does not cope well with sparse data (which is typically the case with events). The latter can make this problem up, but completely forgets the collaborative part which is not acceptable given that events are intrinsically social (we attend an event because other people go, or recommend it). To solve this issue, this paper proposes to recommend events similar to past events liked by people having analogous tastes. It corresponds to an hybrid system with a CB predominance. Unlike [53], no separate recommender module will be developed. Instead, [11] presents a mathematical way to unify CB and CF, based on Perny and Zucker’s previous work an framework introduced in [59]. Modelling of users and items similarities and interactions are performed thanks to fuzzy relations, based on fuzzy sets theory and fuzzy logic. In classic sets theory, an item is or is not member of a set whereas, in fuzzy sets, items have a degree of membership ranged from 0 to 1. This subtlety allows to evaluate the degree of usefulness of items for the active user.

Another very contrasting approach is presented in [19], especially as far as event representation is concerned. This paper introduces CUPID, an hybrid recommender system based on a RDF/OWL ontology modelling of the EventsML-G2 standard [36]. They developed an event aggregator capable of categorizing and enriching the gathered information thanks to Open Linked Data systems such as *OpenCalais* [55], *iKnow* [35], *GeoNames*[25] or *DBpedia* [18]. Events are then distributed through social networks such as *Facebook* on which a recommendation application has been developed. The enriched data is also published in Linked Open Data space, to be easily available through SPARQL queries *via* a SPARQL endpoint.

Less complete but also based on *Facebook*, [37] developed a social-network based event RS. Facebook has, once again, been chosen to create the application. The ver-

sion described in the paper is limited to aggregation of musical events from *Last.fm* [40] but the architecture is designed so that other sources can be added. The main advantage of Last.fm API is the “similarity” function, that returns a similarity distance measure between two given bands and makes recommendation process easier by far. Location information is approximated based on IP. The system does not limit its recommendations to events, as it also recommend friends (i.e. Facebook friends) that can be interested in this event. The philosophy behind this feature is that the user is not likely to attend an event alone.

Using social network is a choice. It can provide extra information about the user and his tastes, as well as it defines a “trust network” of peers whose feedbacks can be considered as relevant for the user (with the assumption friends are more likely to have common way of thinking), making prediction explanation and justification possible. However, it can also limit data collection or freedom for the developer as the RS is embedded into another system and has to obey its rules and use its APIs. Moreover, if the RS adoption rate is not important within a user’s social network, it could be detrimental to the quality of recommendations. A different approach is discussed in [41]: user is asked to grade other users to evaluate their trustworthiness and adapt their respective weights in the CF prediction process.

Finally, due to the location importance in this specific problem, events recommendation in a mobile context have been studied by several research groups [13, 44, 61]. These works try to get the most of modern ubiquitous devices in order to solve the different RSs techniques shortcomings. [61] showed, thanks to mobile phones location data, that people tend to be more interested by events liked by other persons situated in the same location area (typically, the home neighbourhood) than by events situated close to their home. This discovery may have interesting implications in the future of event recommendation for mobile phones. [13] mostly concentrated on how to provide tourists relevant recommendations on their devices by exploiting semantic web, web2.0 and social networks.

Chapter 3

Design and Implementation

3.1 Introduction

As mentioned in chapter 2, a number of calendar events collection already exist over the World Wide Web. Well known websites such as Upcoming, EventBrite, CityLife (for Manchester only) and others are important sources of information for anybody looking for something to do anywhere in the world. Nevertheless, public web-based calendar aggregators (not to be confused with calendar sharing) are still underdeveloped. Calagator is unique in this area as it supports many import formats, from standards to popular websites. Indeed, other actors in calendar aggregation such as Upcoming, ElmCity [22], MashiCal [47] or, before the project died, FuseCal [24] are mostly focussing on iCal format for the input feeds.

Moreover, except from few experiments such as [53] the literature survey presented above showed that events recommendation is also a widely unexplored area. Therefore, this project is unique and innovative. As it was implemented on top of an existing project, the proposed architecture is particular: a web-application part with all the GUI components and pages and an intelligent server part. In this chapter, the architecture will be discussed. Of course, recommendation of events is not very different from recommendation of news articles (see 2.2.3 on page 27) and architectures proposed in literature can be used to implement the system. That is why, for many aspects, the proposed architecture is strongly inspired by the one presented in [56].

Although the architecture is inspired by others, it had to be tailored specifically for Picalag project as each application is different. Functionalities and requirements

for the system are listed in the following section. This chapter then briefly presents the way the project has been organised to achieve its completion (in section 3.3) and the tools used for the development of the system (3.4). The main part of this chapter is the detailed description of the implementation and design choices for all the components of Picalag, in section 3.5. Finally, links to download source code are given in the last section on page 68.

3.2 Requirements Analysis

The following features are required for Picalag.

Users Functionalities

- Users should be able to register on the web-application
- Users should be able to log in and out
- Users should be able to edit personal information
- Users should be able to destroy their account and delete any personal information from databases
- Users should be able to create and import any compatible event and venue in the system
- Users should be able to browse events and venues
- Users should be able to export events to their personal calendar programs or share them on any social network
- Users should be able to search articles and venues by tags (tags cloud)
- Users should be able to search articles and venues thanks to a search box feature
- Registered users should be able to access their personalised dashboard
- Registered users should be able to explicitly rate events
- Registered users should be able to mark a venue as favorite
- Registered users should be able to browse events taking place in their favorite venues
- Registered users profile should capture their interests

System Functionalities

- System should be able to retrieve events from different sources and keep the database updated

- System should minimize automated connection load to event sources when importing data by a reasonable caching system to limit impact on these websites
- System should be able to store the imported or created events in database(s)
- System should be able to extract features from events and manage events vector space
- System should maintain a dictionary of terms used in articles in order to manage events vector space
- System should be able to capture and store implicit and explicit feedbacks from users
- System should update users profiles based on both implicit users interactions and explicit users feedback (on venues and events)
- System should be able to recommend events and venues based on different strategies (non-personalised, content-based, collaborative filtering) using events or venues being browsed and registered users' profiles or ratings
- System should be able to generate recommendations even if the user does not want to provide explicit feedbacks
- System should be implemented as a web-application accessible worldwide through the Internet

Non-Functional Requirements

- System design should anticipate future modifications as much as possible
- System should be implemented in a way that allows scalability
- System should be adapted to whatever computer resource available to run it (at least cross-platform)
- The system is not required to be tested against compatibility with all web browsers
- Response time should be *reasonable*. However, this project is a proof of concept and is not required to be optimized due to time limitation for implementation
- Security is not a main concern for this project and can be improved in future work. Only user's personal data should be stored in a secured database.
- System should be releasable under an open-source license
- System should be fairly documented and code should be commented to be usable and modifiable in the future
- Calagator base software (i.e. except the recommendation part) should be fully functional in case the personalisation server is down or not reachable so that main features (calendar aggregation) are still available

3.3 Project organisation

This section details the deliverables and methodologies adopted for this project.

3.3.1 Deliverables

Several deliverables have been identified:

Picalag web application

This deliverable will contain the calendar aggregator Calagator extended with a recommender system. This application will be delivered as source files and running application installed on a school virtual server (also used during development). Any piece of code produced for this project will be released on a public repository, under open-source license to contribute to community knowledge and to possibly be re-used in other projects (the Calagator developed fork may be pushed to the authors of this project in order to be integrated if they judge the feature is interesting).

It will represent the achievement of this project: a working Personal Calendar Aggregator for events happening in Manchester.

Documentation

The previous deliverable will be explained in a documentation document.

This document will describe user interface as well as some internal features, methods, parameters and interfaces. Installation instructions will also be provided. Thus, it will have two parts: the first one will be a quick user manual and the second one will be contain technical informations and instructions.

Dissertation project report

This report is compulsory for any dissertation project. It is mentioned here just as a reminder: it is part of final deliverables.

This report has to be between 60 and 100 pages long and aims to describe project development and architecture and research results as well as evaluation of the program. It will, among other things, validate or invalidate our research hypothesis.

3.3.2 Research and development methodology

This project will adopt two widely used methodology: constructive research approach and iterative development method. These methodologies are essential for the project to be carried through on time and be relevant for research field.

Constructive research approach

The constructive research approach is presented in [16]. It is a common research method used in computer science and software engineering.

It is typically used to solve practical, real world, domain specific problems by building (here is the constructive part) a theoretical or practical solution. In the process, new theoretical and/or practical knowledge is also created so that the project benefits to the research domain. It is particularly adapted to project as the one presented in this report as it require a project analysis with study of relevant related works, literature and other information sources, the construction of a solution to the problem and, eventually, an evaluation of the proposed solution.

A diagram overview of this methodology can be seen in figure 3.1 on the next page. This diagram has two main parts: (a) “Practical utility” that includes background research, specification, iterative development method that will be detailed in next paragraph of this section and evaluation, and (b) “Epistemic utility” that corresponds to knowledge creation (with the discussion of results and surveys produced in phase (a)) usable by peers to solve new problems in the domain. This method ensures the project will produce novelty and contribute to domain research and be useful for the community (with an open-source public release, for instance).

Iterative development method and software prototyping

The development part, which is an important step in the constructive approach and represents a substantial part of this project will be organised thanks to iterative development method and software prototyping.

Iterative development is now a common standard in software engineering industry to develop efficiently. It consists in several development rounds to add functions gradually. It is also known as “iterative and incremental development”: development phases are organised in implementation cycles (iterative) that include all basic steps of software engineering (plan, specification, analysis and design, implementation, testing,

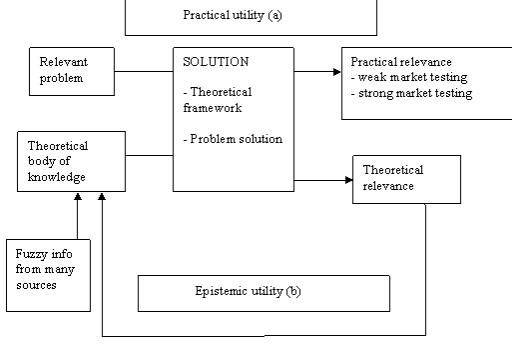


Figure 3.1 – Constructive research approach

Reference Figure 3.1: Constantinescu, Rasvan; *Wikipedia: “Constructive research” article*

evaluation). Each cycle represents a small part of the final system (incremental). This method is summed up in Figure 3.2.

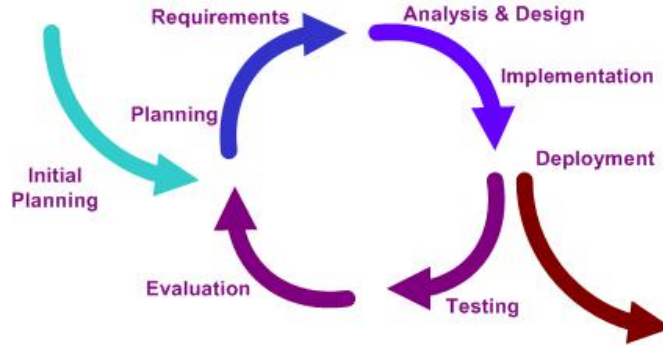


Figure 3.2 – Development cycle diagram

Reference Figure 3.2: Westerhoff; *Wikipedia: “Iterative and incremental development” article*

This method allow the programmers to detect and address errors and the project specification changes early, when it is still not expensive to do so. It also has the advantage to enable a better collaboration with the client or final users. Several implementations exist for the iterative software guidelines. One of them is known as AGILE methods [32] and is particularly adapted to modern development. AGILE is designed to maximise customer satisfaction, thanks to the quick production of useful piece of software based on their needs and expectations.

AGILE is primarily designed to be used by teams of developers. Nevertheless, many principles from AGILE manifesto can be adapted for solo projects [21].

3.4 Tools

This section gives a brief description of tools used for this project. In order to enable community to easily re-use and modify any source code and materials, the development environment tools used are, as far as possible, freewares or licensed under open-source licenses. Moreover, any source code produced for this project will be released on a public repository under open-source license.

Ruby & Ruby on Rails

Ruby is a dynamic, cross-platform, object-oriented programming languages developed since 1995 in Japan. It has been released under Ruby License or GNU General Public License v2 [64]. It has a dynamic (duck typing) type system and an automatic memory management. It is designed to be productive and fun to use, by focusing on how humans would want to program and not how the machine will run.

It is very similar to many programming languages but the syntax mostly comes from a mix of Perl and Python as it was aimed to take the best from many existing languages to create something powerful and strongly object-oriented. The idea is that the learning curve does not have to be a problem for new Ruby programmers that can become productive quickly and have fun.

Ruby also comes with *Interactive Ruby Shell* (IRB), a command-line interpreter.

On top of Ruby, an open-source web framework was built: *Ruby on Rails* (a.k.a. *Rails* or *RoR*) [65]. It has been released under MIT License since 2004 and has contributed to push Ruby to professionals. This framework re-use a part of Ruby philosophy: be productive and have fun. For this reason, it comes with commands to generate many of the most useful parts of a web-application (e.g. scaffolding to generate basic CRUD interfaces).

It implements advanced programming patterns such as the well known Model-View-Controller (MVC) for a better project organisation. It is also designed to emphasize Convention over Configuration (CoC) and Don't Repeat Yourself (DRY) programming concepts for efficient development. The former means that in many cases, no configuration files have to be written as conventions have been stated: this simplifies work at development time, but also ensure that conventions are used to make code reading and software analysis easier whereas it also can cause troubles to the new programmers as they do not necessarily know all conventions and can loose track of

what happens. The latter means that each part of code is located in a unique and unambiguous place.

RoR includes a development web-server, WEBrick but can also be deployed on production web-servers such as Apache HTTP Server (thanks to Phusion Passenger `mod_rails` module). Although this framework proved to be powerful and efficient, Ruby is still an interpreted language and RoR proved to suffer some scalability issues (Twitter used RoR and Matz's Ruby Interpreter but had to partly migrate to Scala, which runs on JVM to satisfy all the requests).

We have to note that the choice of Ruby and the Ruby On Rails framework has been imposed by the project itself as an extension to the existing RoR web-application CALAGATOR is to be developed. However, this framework is raising and is now part the web development industry standards as it is known to be efficient. Moreover, it perfectly fits in this project for which time limitation is a major constraint. Thus, it is probable that even if the web application had to be developed from scratch, this framework would have been a serious candidate and would have been preferred over concurrent PHP (e.g. Zend, Symfony) or JAVA frameworks (e.g. JSF). The major strength of this framework is the simplicity of its native Object-Relationnal Mapping (ORM) layer, which powerfully implements the ActiveRecord design pattern.

RDBMSs: SQLite and MySQL

This project will involve databases. Two famous open-source RDBMSs may be used: SQLite (Public Domain) and MySQL (GNU General Public License).

Both of them are based on the well known and widely used Structured Query Language (SQL). However, they do not work the same way. SQLite is embedded by the program: the database (data, indices, tables...) is stored in a simple unique file (e.g. `*.sqlite` or `*.sqlite3`) that is used through methods contained in a light library included in the program. MySQL engine has a standalone process that use the Client-Server architecture: client sends SQL request then gets the results as an answer from the server.

JAVA

The other major part of the project, the proper “intelligent” logic code, is developed with the well-known programming language *JAVA*.

JAVA language was released in 1995 by Sun Microsystems (now part of Oracle Corporation) under GNU General Public License. It is an object-oriented, class-based, strongly typed language. After development, the application code (*.java text files) is compiled to portable byte code (*.class files that can be grouped in archive *.jar or *.war) that can run on any architecture thanks to the Java Virtual Machine (JVM). It was designed with the motto “write once, run everywhere” in mind and the fact it is cross-platform is one of its major strength. Unless Python, its main competitor (also cross-platform and widely used), JAVA language is not interpreted. This property means performances are claimed to be better (even if the JVM is sometimes considered as heavy). For all these reasons JAVA is nowadays one of the most popular language, especially for web client/server applications, thanks to numerous frameworks.

When ones want to run a JAVA-based web application, a simple and good solution is to implement a special class called Servlet (in case of a web servlet, it extends the HttpServlet Java EE class and conforms to the Java Servlet API). This application extends the capabilities of an application server to address request-response. An HttpServlet runs within a servlet container such as GlassFish (by Sun/Oracle) or TomCat (by Apache). The application server also often manages lots of different useful features, such as database connection pooling.

Several criteria were considered for the choice of a language to implement the Personalization Server (a.k.a. *PServer*):

- The overall system should be coherent (that is to say similar development approach should be used for both the web-application and the PServer).
- If possible, the language should provide useful libraries to use as much existing code as possible to optimize development time by avoiding the “reinvention of the wheel”.
- As described on the background part, RSs are systems of high computational complexity. For this reason, the chosen language should be as efficient as possible.
- Due to time limitation and the fact that a new programming language had already to be learnt from scratch (Ruby) to extend the existing Ruby on Rails Calagator application, the biggest constraint was to use a programming language already known (at least bases).

JAVA is a perfect candidate according to the criteria listed above. Indeed, as Java is widely used, it is often learnt during undergraduate and postgraduate courses, which allows to save a precious time when used in a project as Picalag with time constraint. Moreover, it is a compiled language and performances are there. But the

two first criteria are definitely the most important and the reasons why Java is a good candidate have to be detailed:

Coherence

As the code of this system is to be released, coherence is very important for anyone who wants to participate to the project in the future. Among others, the communication between the different parts of the system should be simple and consistent and similar patterns should be usable in both web-application and intelligent side. The different parts of the application should also be able to run on a single machine (as resources are limited) but scalability should be kept in mind (especially as far as RSs are involved) in case the system is run in a real life production environment.

RoR is based on Representational State Transfer architecture (REST) and setting up RESTful interfaces is made really easy. Servlets can also easily address requests in a RESTful philosophy. By using JAVA with Ruby on Rails, we can ensure simple and clean communication and information transfer thanks to this coherent architecture (see section 3.5).

The Ruby framework uses a powerful design pattern to abstract data from database to ruby objects, called ActiveRecord. It offers an object-relational mapping by wrapping a database table into a class. Especially, the one-to-many and many-to-many relations are converted as object instances of the appropriate types accessible thanks to standard getters and setters. This layer is responsible for all the exchanges with the database (e.g. find, edit, create, delete, save). By default, JAVA does not have any standard ORM. The most popular ORM is hibernate but it is heavy for small projects and configuration can be tedious. Moreover, it is not based on ActiveRecord pattern as it uses DataMapper pattern instead. However, the project ActiveJDBC [2] offers an implementation of Active Record pattern in Java, inspired by Ruby on Rails ActiveRecord. It is released under Apache License 2.0 by an active community and is now mature enough to be used in projects.

Useful libraries availability

In any project, especially when the main goal is a quick prototyping development, it is important to rely on others code. It allows to save a valuable time and to enhance performances and security of the final product, as these other developers focused on the optimisation of their autonomous project more than it could be done if the

development was treated as a small part of a completely different project. The main drawback of libraries is that debugging and quality of documentation is not under control and the number of dependencies is increased. For this project, I consider these drawbacks are largely minimal compared to the benefits.

For Picalag, JAVA offered some very useful libraries. The first one concerns text analysis: when features are extracted from a text, it is important to stem and lemmatize the terms found. For instance, to increase the number of similarities between two text the program has to transform “communities” into “community” or “mice” into “mouse” as it is clear these two words, even if they are different, refer to the same concept. To do so, the library MorphAdorner [1], developed by Northwestern University, has been used.

The second library is the Apache Mahout project [46] which provides collaborative filtering, user and item based recommender systems in an elegant, customizable and simple way. Moreover, it has been designed to use the Apache Hadoop project [30] to ensure the scalability of the algorithms thanks to distributed computing if more resources is needed when the project is used in a production environment.

3.5 Architecture

The global proposed architecture for the system is presented in figure 3.3. This section is a detailed description of the different parts composing the system.

System description

As it was briefly discussed in previous sections, the system is composed of several main parts with different specific roles. The two principal parts are the web-application actually used by user (it can be considered as the front end application) and the personalisation server (a.k.a. *PServer*, which can be considered as the back end application). The third part which can be identified is the automated events import scripts. Each of these top-level component will be described in the following sub-sections.

These parts are linked and can exchange information via RESTful APIs by using simple HTTP requests (mainly thanks to the POST method). It ensures a widely supported communication as any “modern” language has at least an HTTP library

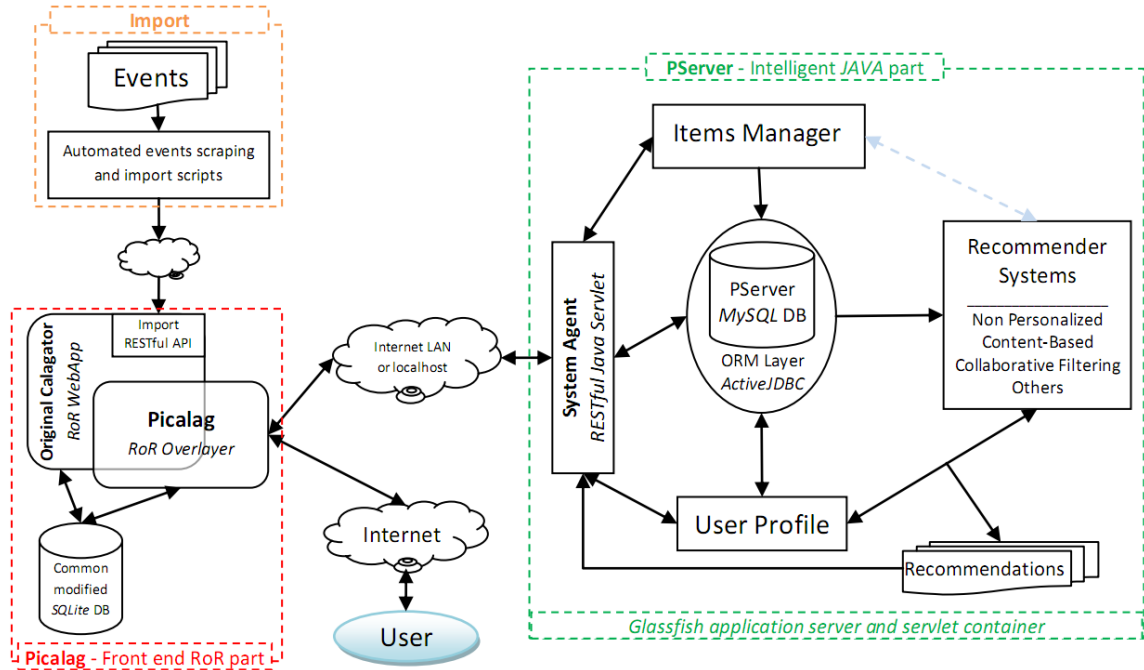


Figure 3.3 – Picalag global architecture

and as this protocol is minimalist and easy to learn. Moreover, this support is important for the coherence in the mean of communication discussed above in the section about JAVA. Complex responses can be generated to simple requests as XML files can naturally be sent over HTTP. In the end, RESTful API is largely lighter than other web-service protocols such as SOAP or XML-RPC as no abstraction layer is added to data that does not necessarily require it. Nevertheless, powerful tools for SOAP and XML-RPC protocols make their use sometimes easier compared to REST (which is not a protocol but a philosophy) that requires to develop own minimalist tools to extract data from the response. All in all, for a modest project as Picalag which runs as a “closed world” coherent whole, it appeared that RESTful was the best choice for communication.

3.5.1 The automated events scraping and import scripts

These scripts are not required for the system to actually do its job but are used to automatically and massively fill the database with new items from various sources (websites and feeds) in a transparent manner from a user point of view. In practice, these scripts will be run periodically (e.g. during the night) by the Operating System they are hosted in to fetch new events from different sources.

The main interest for this module is that it is completely independent from the other parts of the system. They can use whatever existing programming language having HTTP support. Moreover, they can be run in whatever machine able to connect to the front end application (typically any computer connected to the Internet). It means that even if the extraction requires a lot of CPU usage, hard drive space and bandwidth, this process can be entirely uncoupled with the other parts of the system. This design is clearly thought with scalability in mind.

To import the new events in Calagator, a new opened API method was created within the Sources controller. To respect Calagator philosophy, this API is completely unsecured so that anyone can create his own script or program and push events to the system. Indeed, event creation and import does not require any API key or user login (which did not exist in the original Calagator application): Calagator trusts its users, so does Picalag (“We are an all-volunteer effort and encourage anyone to import, create, and edit events” [12]).

This method is meant to import events that are not available in standard importation formats supported by the initial Calagator application. In this respect, it contributes to make Calagator even more universal and capable of import any event. Indeed, parameters of the POST method will be used to fill the different fields for the new event/venue just like if the form to create a new event was manually submitted. For any already supported formats, the original import action from the Sources controller can also be used to automatically post the URL to the event to import.

3.5.2 The front end web-application

The front end application is the proper web-application the user will interact with. It is composed of two main parts:

- the original Calagator application slightly modified to integrate recommendations and user interaction GUI
- the Picalag specific pages and components added for this project with the new features

From now on, the term “*Picalag*” will refer to either the whole system or the whole front end (as opposed to “*PServer*”) in this Dissertation, for clarity reasons. Precisions will be provided in case of ambiguity.

Both part use a common SQLite3 database. Tables were added to the original database to be able to store user’s login information. The choice of this RDBMS

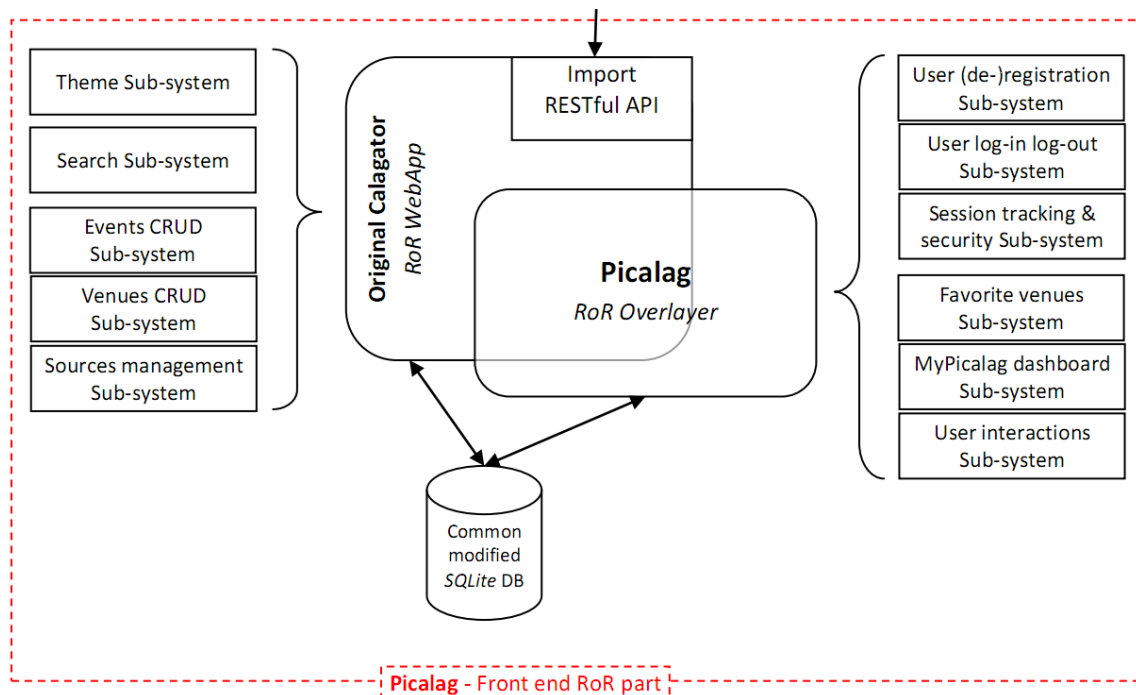


Figure 3.4 – Picalag front end architecture

was imposed as Calagator original application was already using it. Besides, SQLite is used by default in any Ruby on Rails project (it then can be configured to use another database system). Even if SQLite looks simple (it is a single file and the logic code is in a language specific lightweight library) and does not have many features (e.g. no user control: the OS is responsible for read/write rights), it claims to have decent performances. For this reason, it is far enough for this type of project in which security is not the main concern and size and traffic are expected to be modest. If SQLite shows some weaknesses in a production environment, migration to MySQL or another RDBMS is still possible without any change in the code as Ruby on Rails uses an ORM abstraction layer (based on ActiveRecord pattern).

A more detailed diagram of this part of the system is proposed in figure 3.4. It shows the whole application (i.e. including the two main parts described above) can be split up into 11 sub-systems that more or less correspond to the different controllers present in the MVC pattern of the application. Each of them is responsible for specific actions of the systems.

Theme sub-system

This sub-system is responsible for the global rendering of the application. It contains Cascading Style Sheets (CSS), javascript files and site layout. Some partial views are also located in this sub-system such as site description and presentation.

For Picalag, most of the original theme could be reused. However, the layout (containing the header and top menu) and CSS files were modified to integrate new menu entries to access new features and the log-in form accessible from any pages of the website.

A settings file (settings.yml, formatted according to YAML [YAML Ain't Markup Language] standards) in this module is responsible for site name and title, time-zone to use (London), GPS coordinates (used to center the GoogleMaps map on Manchester) and URL address to the intelligent part of the system (PServer). These settings are loaded when the server starts and are available in a global variable from anywhere in the application.

Search sub-system

This sub-system helps users to search events and venues stored in the system thanks to two search forms.

The “search venue” field is printed on “Browse venue” webpage. It uses a simple SQL request to search **keyword** in venue title or description.

The “search events” field is printed on top right corner of any pages of the website. This form is more developed as it can work according to different modes:

- Default mode, using a simple SQL request to search **keyword** in event or venue title and description. It provides sub-string matches. This module was adapted for bigger datasets in Picalag as it was too slow in its original version. It now searches events only for a given date.
- Sunspot mode, using the java-based sunspot search engine. It requires extra setup and `gem`¹ dependencies but provides relevance-based sorting (in the other hand, sub-string matches are lost).
- `acts_as_solr`, using java-based Apache Solr search engine that provides both relevance-based sorting and substring matching but sometimes causes database records creation and edition to slow down.

1. RubyGem is Ruby's packaging system. Packages are called gems. It is used to install dependencies for a ruby application.

Event CRUD sub-system and Venue CRUD sub-systems

These sub-systems are responsible for the basic CRUD functions on events and venues records. *CRUD* corresponds to the four basic operations on data: *Create*, *Read*, *Update* and *Delete*.

Moreover, these sub-systems were modified from Calagator to Picalag to ensure the front and back end databases are synchronised: when an event or a venue is created, updated or deleted from the front end application, an HTTP request is sent to the back end PServer to make it reflect the modifications in its own database.

Sources management sub-system

This sub-system is responsible for events and venues importation from various sources. The “Import RESTful API” described above in 3.5.1 and represented in figure 3.4 is also part of this sub-system (it was only drawn in an independent box to show interactions coming from outside the application).

When a user wants to import an event from a source with the supported format (using the import form), this sub-system is in charge of parsing the source and extract the available data to create or update the event/venue. When the API method is used, each data is sent in a specific variable as POST parameters and used to create or update the event/venue.

User (de-)registration sub-system

It is responsible for the collection of required information to create a new user account. It is also in charge of deleting existing account and user’s personal information if, for any reason, the user decides to quit the system. In both case, this sub-system ensures an HTTP request is sent to the PServer part of the system to keep both database up-to-date and synchronized.

Among others, this sub-system is in charge of password encryption before storage and minimal security (using salt to avoid rainbow-table decryption attacks in case the database is stolen).

User Log-in Log-out and Session tracking & security sub-systems

These sub-systems are responsible for the authentication and the security of the system. They do not need to communicate with PServer. Log-in form is used to

authenticate user thanks to his login/password, then session tracking keep the authentication opened for this user.

The security sub-systems also filters pages to make sure nobody can access protected actions if not logged in to prevent information from being stolen or modified and system to crash because of illegal actions.

These subsystems, as well as the User (de-)registration sub-system are developed using a light, simple and model-based authentication Ruby on Rails plugin: Authlogic[5].

Favorite venues sub-system

The task of this sub-system is to manage any actions related to favorite venues list.

When a user is logged in, he can access a webpage that shows a list of the venues he stared and the events happening in these venues if any. This is justified as people tend to often spend their leisure time in a small number of venues. Being able to follow what is on these venues is an interesting feature as it corresponds to natural, cheap (in term of computation) and accurate recommendations. Moreover, it is easier to express a rating about a venue than a future event. Least but not last, venues are records with a low churn (i.e. few new venues are added to the system everyday and the old records are not perishable as are events) and preference for users regarding a venue is stable in time. All these properties make the use of venues for recommendations easier than the use of events only, provided that the user plays his part of the game and explicitly stars venues.

On the same webpage, the system propose venues recommendations. These recommendations are of two different types: most popular venues (based on the number of users who “like” a venue) which corresponds to non-personalised recommendations and collaborative filtering item-based recommendations (with Mahout library). The later are computed in a “people who liked these venues also liked” style and each recommendation is justified thanks to a list of venues (contained in user’s favorites) they are similar to. The purpose of these two recommendations method is to broaden user’s venues list by making him discover new venues he may like.

The last role of this sub-system is to manage “add venue to favorites/remove venue from favorites” widget on an event or a venue page if the user is logged-in on the right panel (see figure 3.5). This AJAX widget is used to manage favorite venues list. When the page is loaded, the current status for the venue is requested to PServer and a link is generated to add or remove from favorites. If the user clicks this link,

a request is sent to PServer to update venue status, then new status is fetched to refresh the widget. This provides a dynamic immediate visual feedback and enhances user’s control over the system as well as his global experience as he does not need to wait the whole page to be refreshed.

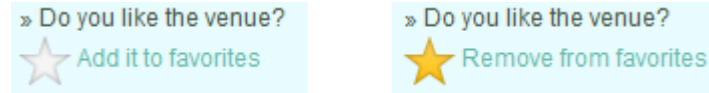


Figure 3.5 – Manage favorite venues list widget - even page (before/after click)

MyPicalag dashboard sub-system

This module is the central page for the registered user to get all recommended events. Four different types of recommendations are proposed:

- Most popular events: according to the cultural events recommendation case study presented in [53], non-personalised popular events list tends to be the recommendation type the users trust the most (therefore, they are the most clicked links). It is also cheap to compute them. Thus, it is important to include this list in a recommender system as the user probably would not understand (and forgive) if this basic and simple feature is not implemented. These recommendations are sorted and justified according to the sum of all their ratings.
- Random events: this non-personalised list is also nearly free to generate in term of CPU time. It has two main interests. Firstly, it can be appreciated by users as it can make them discover new kind of events they even would not think about and broaden their tastes and interest. Secondly it can be used as a neutral base for statistical tests to verify if the click behaviour is really different from one list of recommendation to another. Indeed, if click count is not statistically different from random, it can be inferred that recommendations are not statistically different from random (i.e. they are not really useful recommendations).
- Content-based recommendations: they are generated from an hybrid recommender system (mostly content-based). It could also be called “Similar users go” as the RS is comparing events vectors (initialized as *tf.idf* but updated with users profiles features each time they are browsed) with user’s profile (updated with browsed events features). These recommendations are justified with their relative match (in %) to user’s profile as computed with the cosine similarity distance by the recommender system.

- Collaborative filtering recommendations: they come from a Mahout user-based recommender algorithm with the Pearson similarity distance described in chapter 2. The recommendations correspond to what people with similar taste think about these events. Each recommendation is justified with the average rating given by user's neighbourhood.

This sub-system is also responsible to fetch and print recommendations on show event page:

- Content-based recommendations: similar events based on their *tf.idf* weighted features vector justified by the cosine distance match to the browsed event (in %).
- People who viewed this event also viewed: these recommendations are computed thanks to an item-based mahout recommender system. This RS returns item-item recommendations: it gives the most similar items based on users' ratings only (using the log likelihood similarity distance)

Each of these widgets are initialized in AJAX when the page is loaded to show 5 records. The user can then click a link on each widget to refresh it individually and show more records if any (+5 records for each click). The use of AJAX is justified here as each recommendations does not take the same time to be computed (e.g. most popular and random are immediate while CB or CF require more computation time). With this choice, the first recommendations are printed immediately when the others are still being computed. User waiting time are less important as the page comes instantly then the recommendations come gradually when they are ready to be printed. It also increases user experience as each widget can be refreshed individually.

User interactions sub-system

When a record is read by a registered user (i.e. the user browsed a particular event or venue), a request is sent to the PServer in order to trigger the methods and update user's profile and record's representation on PServer side. This allows PServer to adapt user's profile thanks to implicit rating.

Other implicit rating operations are triggered using AJAX when:

- the user adds the event to his calendar (i.e. he uses one of the export functions of calagator, under the responsibility of Events CRUD sub-system).
- the user uses the share menu to publish the event to one of the social networks he subscribed (this functionality has been added to the original calagator application as it was surprisingly missing). The feature is added thanks to the AddToAny javascript plugin [3] and the AJAX method is called thanks to a callback function.

We consider that if somebody uses these actions, it means he likes the event. This is justified as he is probably setting a reminder up or looking for people to attend the event with him. As a consequence, the best grade is used for any of these two actions and the profile is updated to take this interest into account. This mechanism allows to collect implicit rating in a transparent manner for the user as he just has to do normal actions.

This module also offers a way to give explicit feedback for each event in a three grades scale: like/no opinion/dislike. Compared to richer scales (from 0 to 10 for instance), a three grades scale is less disturbing for the user who does not necessarily have enough details or knowledge to make a real difference between a 7 or a 8. When users clicks “like”, the highest grade is applied, when he clicks “no opinion” the same grade as “viewed” is applied and when he clicks “dislike” the profile is updated to diminish the weight of the event’s features (then similar event are less likely to be recommended). The widget uses AJAX and gives immediate visual feedback of the grade (see figure 3.6).

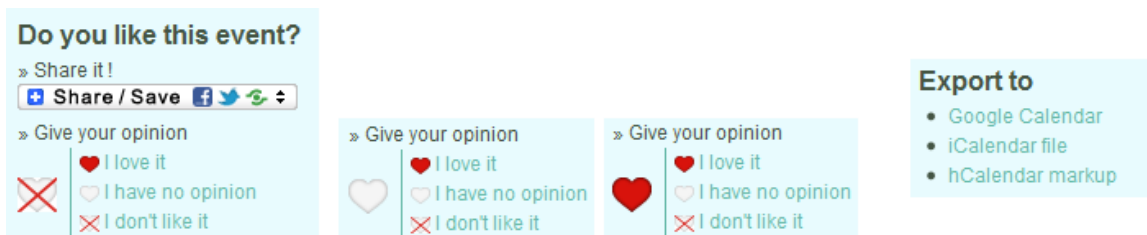


Figure 3.6 – Event rating widget, share widget and export function

The last role of this module, during the evaluation period, is to log user interactions when a recommended record link is clicked (from MyPicalag dashboard, favorite venues page or an event page). These log entries will be used in following chapters to extract statistical data.

3.5.3 The PServer back end application

The architecture of the PServer JAVA back end was already sketched in the global architecture diagram proposed in figure 3.3 on page 46. However, this section will detail each module separately as this part of the system is more technical and complex than a simple GUI front end website. It really represents the “brain” of the system as it is in charged of computing and returning the recommendations. PServer is also

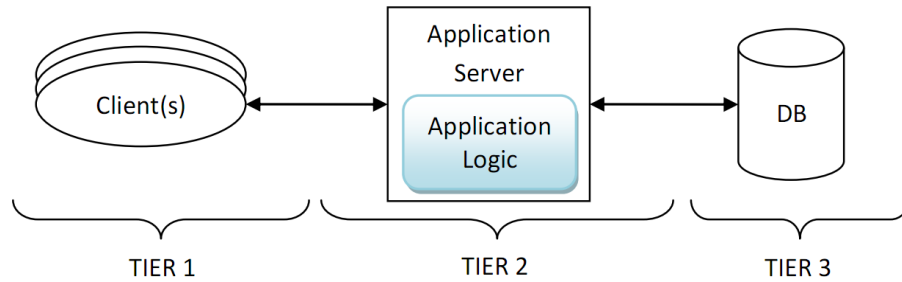


Figure 3.7 – Three-tiers model

the most demanding part of the system in term of resources and CPU time. Indeed, all machine learning and RSs algorithms are concentrated in this server.

PServer is built as a completely independent web-server application. Again, just as it is the case for the import scripts and the front end, this design allows the administrator to install this part of the system on a separated machine with dedicated resources. The use of JAVA and Servlet Java EE standards make its deployment easy on any machine able to run Java SDK JVM, a GlassFish (or any other) application server and any RDBMS compatible with ActiveJDBC library (support is currently limited to MySQL, PostgreSQL, Oracle or H2). When the application is compiled, it is packed with all required libraries in a unique ~20MB *.war archive file, ready for deployment. The RDBMS can possibly be a database running on a distant server. All these properties were chosen to maximize the scalability of the system by making each part requiring resources independent. Moreover, server migration is painless: if PServer URL changes, the administrator only has to modify one line in a settings file of the front end RoR application (see theme sub-system subsection above).

As the front end, PServer uses the classic three-tiers model by default as described in figure 3.7. The only difference is that PServer is designed to serve a unique client, Picalag front end, whereas the Ruby on Rails application is made to serve multiple users as well as the import scripts. We do not think PServer should be able to serve multiple picalag instances, mainly for performance reasons.

Database access

By default, PServer relies on a MySQL RDBMS server for the storage of any dynamic data. As we discussed, the abstraction layer ActiveJDBC offers the opportunity to add flexibility by supporting different other RDBMSs without requiring a single change in the code. Indeed, this ORM converts database records into JAVA Model

objects and any read or write access to the database are managed through these objects thanks to getters, setters and methods.

The choice of MySQL was not based on performance considerations. The only reasons it has been used in this project is that it is one of the easiest server to set up (many turnkey solutions can be found on the web e.g. Xampp [78]), widely spread thus well known and well documented and it is open-source (maintained by Sun, now Oracle Corp.). Often, MySQL is installed with web-server Linux distributions and is running out of the box. This choice was then natural and imposed itself.

For the database connection management, however, performance has been taken into account. To limit the cost of opening, maintaining and closing database connections, we opted for the use of connection pooling. GlassFish server natively offers connection pooling feature. After a quick configuration from the GlassFish admin panel, a cache of maintained connection is created so that they can be reused several times without the need of opening a new one, then closing it when the job is done. The application server manages any connections and the application does not need to bother about it. To pass the data source to the application, a JNDI name is given to the JDBC resource (in other words the connection pool with a name like “jdbc/picalag_pserver”).

To pass the connection to the application, the resource should be passed through the application context. As we are using ActiveJDBC ORM layer, the Base object of this library should be the one which gets the connection. To make sure connection is ready when PServer Servlet code starts processing the request and to close the Base object in a gentle way once work is finished, a servlet filter is implemented. Filters are Java classes that implements Filter class. They can process request and response before (pre-filter) or after (post-filter) the servlet. This allows to set a chain of filters after and before the servlet. Each stage of the processing does not need to bother about what is done before and after it. This can for instance be used to initialize objects (as our ActiveJDBC base), or transform the response (e.g. an XML response transformed into HTML thanks to an XSLT processor filter). In the case of PServer, a pre-filter is used (but it’s role continues after the servlet generated the response to close the Base object). To illustrate what happens internally when a client sends a request to PServer web-application, a diagram is proposed in figure 3.8.

In PServer web-application *web.xml* config file, which specifies the settings the container should use when the application is deployed, few lines are added so that GlassFish knows that a pre-filter has to get the request and that it needs the JDBC

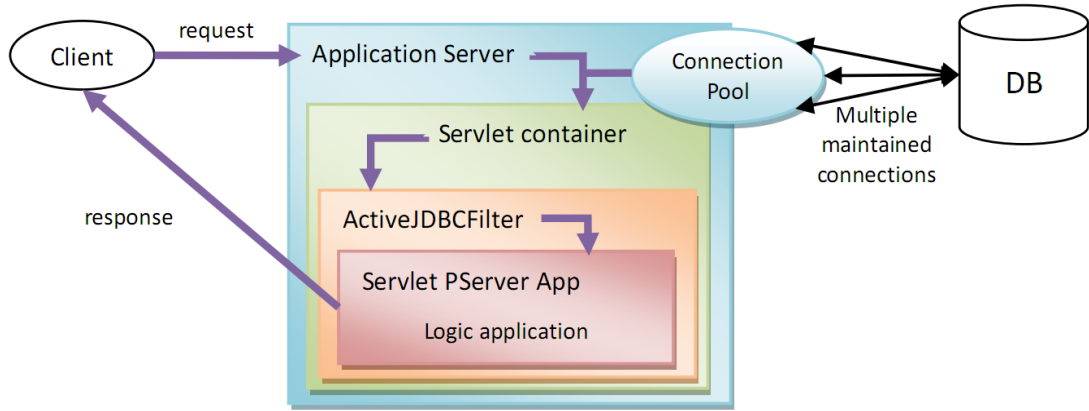


Figure 3.8 – PServer request processing path

resource as shown in figure 3.9.

```

<filter>
  <description>This Filter opens an active JDBC connection to a database from jndi (connection pooling)</description>
  <filter-name>ActiveJDBCFilter</filter-name>
  <filter-class>pserver.ActiveJDBCFilter</filter-class>
  <init-param>
    <description>connection pooling jndi</description>
    <param-name>dbConnection</param-name>
    <param-value>jdbc/picalag_pserver</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>ActiveJDBCFilter</filter-name>
  <servlet-name>PicalagPserverServlet</servlet-name>
</filter-mapping>

```

Figure 3.9 – PServer web.xml config file

Database design

The database structure is reproduced in figure 3.10 as a UML model. It is composed of 9 SQL tables that store the data required by PServer to generate recommendations. This schema will not be extensively discussed as the diagram is clear enough to understand the role of each table. Only some details have to be explained.

Firstly, an event has two types of features that compose its vector: boolean features (i.e. the event has this property, or it does not) which basically correspond to tags identified by the system when the event was imported and weighted features (used to evaluate the weight of a feature in the event's identity). Weighted features are initialized as TF-IDF weights based on description, title and venue name of the event (frequency for each feature in the event is also stored to be able to compute tf-idf weight later on) then evolve as the event is browsed by users to store users interest and be recommended to other similar users. Boolean features are set when the event is added and are then static. Choice was made to store these two different

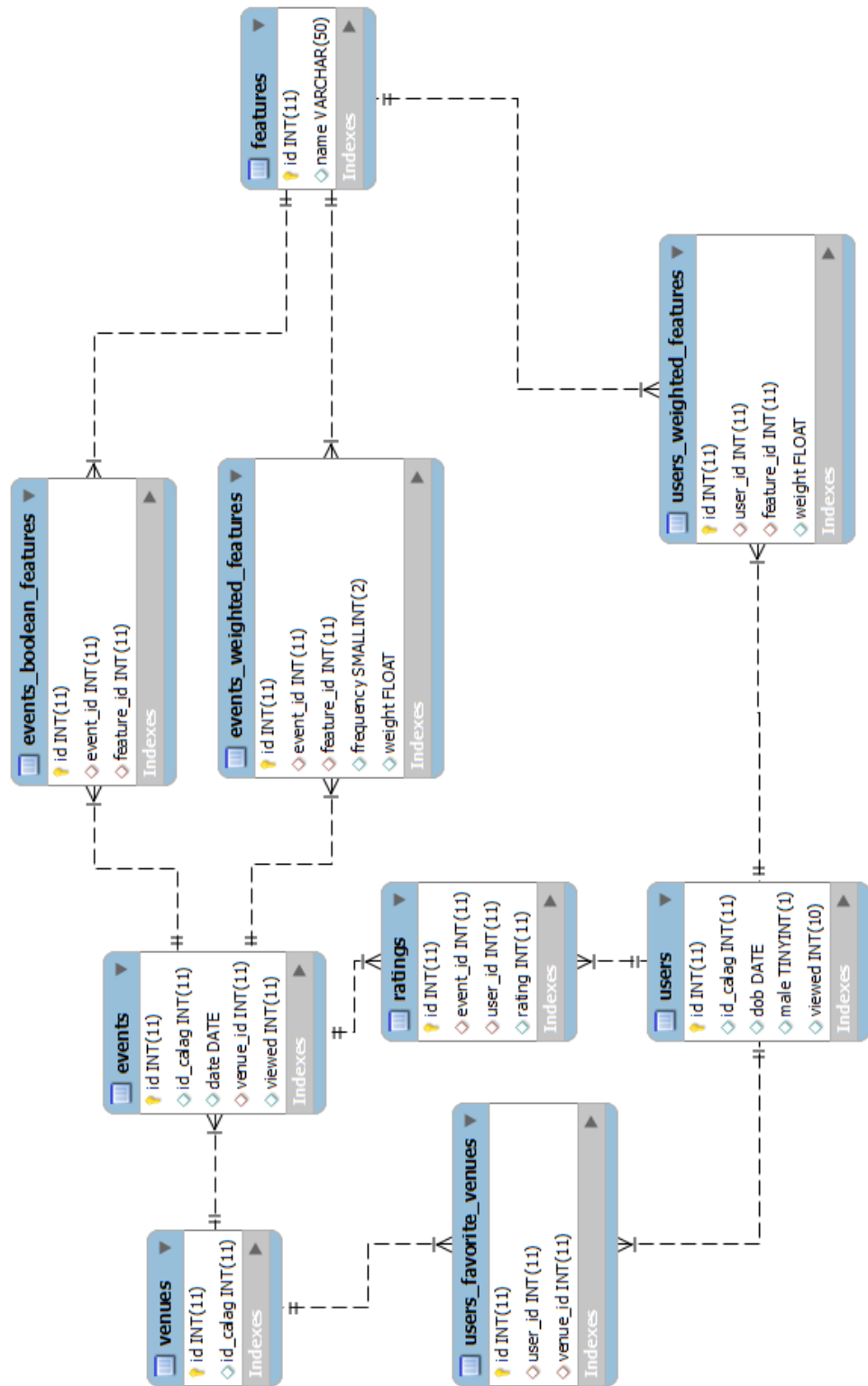


Figure 3.10 – PServer database diagram

types of vector components in different tables rather than in a unique table with a flag field to be easily able to exploit them in different model objects in our program (see ORM description above). Indeed, even if they are both part of a feature list, their role and actions are slightly different. Moreover, we think that this structure choice makes the system more flexible for future modifications or to add new features' behaviour or role.

Secondly, user profile is only composed of one type of feature: weighted features. This is justified as the system should be able to adapt it according to each user interaction action with the front end. As a consequence, no part should be static to be able to capture shifts in interests for instance.

Note that uniqueness constraint is enforced for “name” field in features table so that processing can equally use strings (name of feature) or id.

Another point is that the “venues” table is nearly empty and could probably have been deleted. Again, as it is the case for the different types of event features, this choice was made to have a Venue model in the java program. Furthermore, the system does not fully exploit possibilities about venue in recommendation computing or profiles updating. It could be for example improved in future works by taking venue description features into account. As the system was designed to be re-used, customized and improved, it appeared logical to prepare future modifications and make them easier by having a specific table ready. Indeed, the less future developers have to change the database structure, the better.

The “users” table stores information about date of birth and gender of the user as given during the registration process. This can be used to initialise user's profile vector after registration (see demographic RS page 25) based on the example of [52] to partly solve the new user problem. However, this feature was not implemented in this version of PServer and user profile is initialized as empty.

Finally, Venues, Users and Events tables have an “id_calag” field. This field is made to store the corresponding object's id in the front end. It is managed so that when a request is received (with ids from front-end database) the system can find the correct object, computations can be internally made with the pserver id (imposed by Active-JDBC and for data integrity purposes) then the response makes the correspondence again to return ids of items as they are stored in the front-end database.

PServer servlet / System agent sub-system

The Servlet is the central module of PServer. It is responsible for receiving request and issuing the response. It can be considered as System agent as it processes users' interactions with the system through requests sent by Picalag RoR front end. The servlet has several roles:

- it receives the request and extract the parameters attached to it if any
- it implements the RESTful API methods so it must be able to route the request to the correct method. If no RESTful method corresponds to the request, it fires an error 404 “method not found” response.
- it checks if arguments for method are valid. If not, it fires an error 400 “missing arguments” response.
- it calls the corresponding methods to execute the action requested by the client.
- it finally gets the returned objects from previously called logic methods if any and process them to a response that can be either an XML or plain text. Response is generated thanks to either a JSP page, or the output stream. In some cases a simple blank #200 “OK” status response is sent. If something is wrong during the processing, an error 500 “internal server error” can be fired.

Items manager sub-system

This module is mostly used when a new event or a new venue is added to the system. The case of a user adding a new venue is trivial as PServer does not do much about venues. That is the reason why this action does not require a particular processing. However, when a new event is added, the system has to analyse the textual content and process the data about it. This is the role of Items Manager sub-system.

To complete its mission, this module has a main class called EventAnalyser, which then uses different tools as it is represented in figure 3.11. These tools are part of this sub-system as they are used to automatically process new raw items but they could be used by any other part of the system if required.

The first thing to do when a natural language text is processed is to clean it in order to make sure only the relevant information contained in it is kept. This is the role of the Stop Words Manager tool. “Stop words” are filtered words as they are considered as not relevant in a text (e.g. “the”, “as”, “is”, “this”). Lists are available on several websites but there is not a definitive stop words list. Of course, a short list leaves too many useless words and a large list risks to erase important information. Thanks

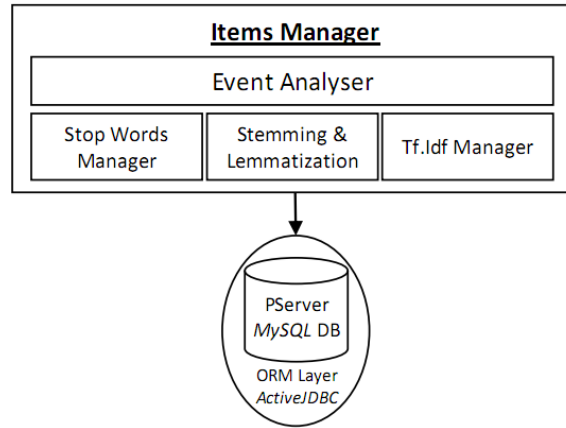


Figure 3.11 – Items Manager sub-system

to regular expressions, the text is cleaned from these words, punctuation marks and meaningless digits.

Once the text is “cleaned”, the second step to maximise the relevant information extraction is to stem or lemmatize the words. This is the role of the Stemming & Lemmatization MorphAdorner library [1]. First, the text is tokenized (we split the text on space character). Then, each token is lemmatized thanks to the library algorithm, to be able to group together the different forms of a each word and analyse them as a single word. For instance, the words “child”, “children” and “childish” are lemmatized as “child” (the stem form) so that items with either one form or an other will have a unique final feature: “child”. This step is important to limit the number of different features and increase the overlapping of items’ features vectors (therefore it also improves the detection of similitude between them).

However, it is essential to notice that this module has important limits. Firstly, it does not solve nor synonymy, nor homonymy classic problems (e.g. a “concert” will not have any link with “live”). Moreover, the lemmatization is not always accurate and this stemming process can create homonymy (if the computed stem is the same, even if words were different). Secondly, multi-words concepts are not captured (e.g. “day out” will become two features “day” and “out”, and “brass band” music style will also be split as “brass” and “band”). These problems are often faced. These limitations could be solved using an advanced semantic analysis and ontologies, however these tools are still under active research. We considered these drawbacks as acceptable for this project given that semantic processing is complex and was not a core problem for us.

The third tool which was developed for this sub-system in order to analyse new

items is the Tf-Idf manager module. This module computes weights for each identified feature of an item. The computation is based on formulas introduced in the background chapter, when content-based RSs were described (sub-section 2.2.2 on page 19). The weighted vector output by this tool is used both to initialize a new event’s weighted features vector and directly by content-based recommender system. In the latter case, it is used to recommend events similar to a browsed event. Indeed, weights are evolving as users browse the events, and features are added that were not found in event’s description or title. This module then returns only the initial features with their tf-idf weights to find similar events. During tests, results were empirically found to be more accurate.

Originally, a fourth module was implemented and empirically tested to enrich the features vector of every newly added event. The aim was to increase the vectors overlap with other events’. To do so, a simple Mahout recommender system based on the Slope One algorithm [42] was used to suggest suitable features to add to an event vector, based on all feature-event associations in the database (i.e. events were considered as users and features as items, the weight was considered as the rating). Even if this module was left in the code as it could receive further improvements in a future version (e.g. with filters, rescorers or improved data model), it is not used in the current system as the benefit was not found to be substantial as is. This is the reason why the module does not appear in figure 3.11.

User profile sub-system

User profile sub-system is responsible for transforming any action performed by the user interacting with the front end application into an action on his virtual profile on PServer side. In other words, this component can be considered as the PServer mirror of the “User interaction sub-system” on Picalag web-application side (see page 53). To be able to handle the interactions given to the user, this sub-system has to be composed of three modules as suggested in figure 3.12.

Of these three modules, the most important one is without any doubt the profile manager. This part has to update the profile vector whenever the user performs any action about an item (event). Formulas used to update the vector are those described in[53]. When a user u visits an event (e) page, the system updates his profile (represented by the weighted features vector \mathbf{w}_u , updated as the new vector \mathbf{w}'_u) as

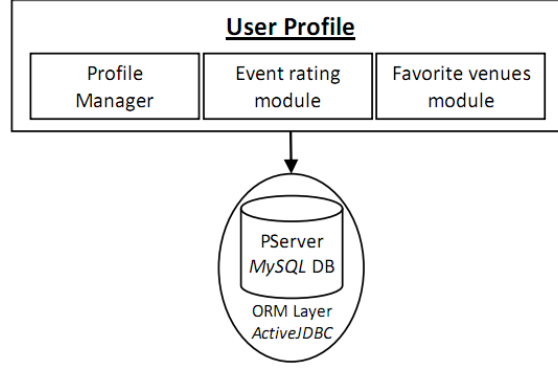


Figure 3.12 – User profile sub-system

$$\mathbf{w}'_u = \frac{N_u \mathbf{w}_u + \tilde{\mathbf{w}}_e}{N_u + 1} \quad (3.1)$$

where N_u is the number of events previously browsed by user u and $\tilde{\mathbf{w}}_e$ represents the event vector (including both boolean features and weighted features vectors) which components are chosen as:

$$(\tilde{\mathbf{w}}_e)_f = \begin{cases} a & \text{if feature } f \text{ is in } e\text{'s boolean features list only} \\ \max(a, w_e^f) & \text{if feature } f \text{ is in } e\text{'s weighted features list with weight } w_e^f \\ 0 & \text{otherwise} \end{cases}$$

(in this formula, $a = \text{avg}_f(w_e^f)$ represents the average weight of all weighted features for event e). Then, with similar notations (N_e is the number of visits of the event's page), event's weighted features vector \mathbf{w}_e is updated thanks to the newly updated user's profile \mathbf{w}_u as:

$$\mathbf{w}'_e = \frac{N_e \mathbf{w}_e + \mathbf{w}_u}{N_e + 1} \quad (3.2)$$

Profile manager module works in close collaboration with the rating module. As its name implies, this second module is responsible for handling rating interactions. Indeed, as we said in “user interactions” front end sub-system section, when user visits an event for the first time, rating is implicitly turned from 0 (neutral) to 1 in the same time his profile is updated as described above. User can also implicitly (e.g. he exports the event to his personal calendar or shares it on a social network) or explicitly rate the event which leads to a new profile update:

- if the rate is “liked”, a coefficient 2 is applied to $\tilde{\mathbf{w}}_e$ in equation 3.1
- if the rate is “disliked”, user profile should be updated to weakened weights of features which describe the disliked event:

$$(\mathbf{w}'_u)_f = \mathbf{w}'^f_u = \max \left\{ 0, \left[\mathbf{w}^f_u - 2 \cdot \max \left(1, \frac{N_e}{N_u} \right) \cdot \mathbf{w}^f_e \right] \cdot C \right\} \quad (3.3)$$

where f is a feature appearing both in user’s profile features vector \mathbf{w}_u and the weighted features vector of event e , and

$$C = \begin{cases} 0.5 & \text{if } f \text{ is an initial feature of } e \text{ (i.e. is present in event's text)} \\ 1 & \text{otherwise} \end{cases}$$

Thanks to these two essential modules, the system is able to collect and maintain a vectorial representation user’s interest (equations 3.1 and 3.3), and capture the profiles of people interested in each event (formula 3.2).

The last part of this sub-system manages the favorite venues list of each user. This module is straight forward and does not need further explanations. Note that this module is also used to fetch the whole venue list when a user requests it (see favorite venues sub-system in the front end application on page 51).

To sum up, this sub-system consists in a virtual agent for each user. The three modules it is made of are able to handle and maintain any data and interactions with the system. User features vector, ratings and favorite venues list compose the user profile. The quality of this virtual agent is a crucial factor that determines the quality of recommendations generated by the system as it is responsible for the data used to feed the recommender systems algorithms.

Recommender systems sub-system

The last piece of PServer is, of course, the recommender systems sub-system. This part is the *raison d’être* of the whole system as all the other sub-systems are created to feed it with suitable input data and are useless without it (in the sense they do not bring anything to the user). Recommender systems are the engine where the value added is created. This module is composed of many different parts, mostly based on the different types of recommender systems introduced in chapter 2, as sketched in figure 3.13.

First of all, a distinction has to be made between the various recommender systems

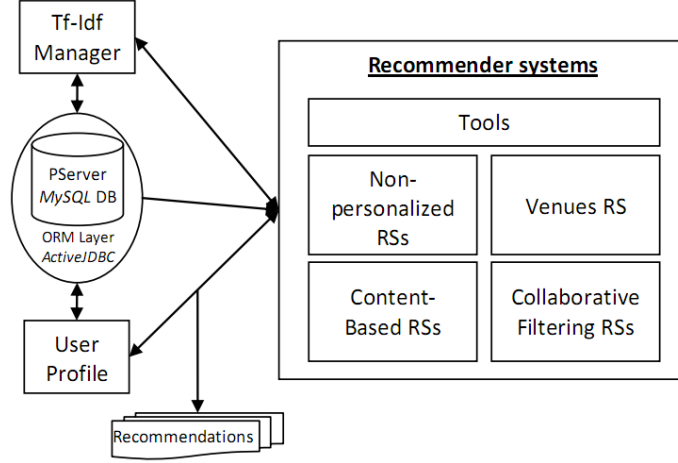


Figure 3.13 – Recommender systems sub-system

themselves and reusable tools that were developed to support them and are required in their algorithm. This first part is a toolbox to solve diverse problems such as:

- Computation of the *cosine distance* between two events of an event and a user's profile (for the formula see 2.1 on page 20).

This distance requires a processing to compose events and users vectors corresponding to these resources. It can include information such as weighted features, boolean features, venues id, favorite venues, tf.idf weights. Therefore, this tool logically depends on the previously introduced TfIdf manager module and the whole User profile sub-system to do its job.

- Filtering to indicate to the recommender systems what items are recommendable in each case.

These filters implement an Apache Mahout interface called IDRescorer. This interface describes classes used to rescore and filter objects during the recommendation process. In PServer, the rescoring part is not used but recommendable items are filters (thanks to a simple set of ids) to recommend only events happening at a given date (as the choice was made for performance and relevance reasons that recommendations are always generated for a given calendar day). In a future development, new rescorsers could be developed that could give a higher score to items close to user's profile and a lower score to items close to user's disliked items. This would constitute a relatively cheap (in term of efforts) hybrid system.

- Miscellaneous problems.

These tools are not categorized as precisely as the previous ones as their roles are less important and perspective of future work and improvements are less interesting. For instance, representation classes such as recommended events or recommended venues

can be found in this category. Both of them represent a recommended item with the reason(s) why it is recommended by the system if any (e.g. the distance to an event or a profile or the number of fans for a venue). An other example is a class representing an ordered hashmap allowing to keep only the top N items according to a numerical parameter and comparator classes used to sort it based on rating, distance or number of fans. Typically, these tools are used to solve the communication problem between the recommender systems which generate a list of recommendations and the system agent that transforms this list into an XML response (a la middleware).

With all these classes, plus the Apache Mahout java library, four main types of recommender systems have been built for this project. All of them use items vectors (stored in database or generated by the Tf-Idf module of Items Manager sub-system) and/or users profiles (including ratings) as input to generate the recommendations. In this version of Picalag PServer, recommendations are computed on demand. The other solution would have been to generate the recommendations when possible and store them in a database for quicker retrieval. For this proof of concept system it was not found crucial to spend time on such a cache implementation. However, as recommender systems suffer scaling issues and as the number of users grows, it could be interesting and easy to work on this for a future version of the system. The recommender systems were implemented so that it is easy to use them independently of any request and we can imagine the recommendations could be generated periodically or whenever parameters change in the system (e.g. an event is rated) then updated into user's profile (with a new dedicated module). During the evaluation of the system (see chapter 4), diverse performance indicators such as the response time have been logged to confirm or invalidate the need of a cache for recommendations.

The different implementations and outputs associated to each of them were roughly described in the front end presentation on page 52. The name of each recommender system is clear enough for the reader to associate which system is responsible for which recommendations.

Mahout recommender systems are used in Collaborative filtering and venues recommender. These systems are claimed to be highly scalable as they are built on top of Apache Hadoop library which allows to set up distributed computing algorithms. The used recommender systems were not modified (except for the filtering based on calendar days). Results could be highly improved by the use of customized data models, adapted similarity distance, fine rescorsers (e.g. to filter out events close to disliked events and makes the score of possibly liked events higher) and advanced

filters. Mahout propose well documented interfaces and some tutorials are available on the internet. However, this would have required too much time for this project and the efforts were mainly focused on content-based/hybrid recommender system that was developed from scratch. There is no absolute answer to “what algorithm would be the best?” and “what distance should I use?” as it is highly depending on the data. Mahout library offers diverse ways and metrics to evaluate the relevance of a type of recommender and a similarity distance over a dataset. The choice of the best implementation would require lots of testing on different datasets and could be done in future works.

Another possible point of improvement comes from the way the data is fetched to build the data model used by the recommender system. At the moment, data is selected directly from the MySQL database thanks to MySQLJDBCDataModel class. This method is claimed to be a problem for database performances as it does not cache the data in RAM and thus requires lots of requests (i.e. $O(n^2)$ where n is the number of users/items) and might slow the recommendation generation process down. Therefore, this way of doing is not adapted for large datasets. Several possibilities are offered to limit database access that could simply be implemented in further developments:

- if it is not a problem to load the whole dataset in RAM, a temporary csv file could be created with the data (e.g. “user_id,item_id,rating”) then parsed thanks the FileDataModel class. In this case it might be interesting to keep the DataModel and the Recommender objects alive somewhere and only refresh them whenever the system feels it is required (i.e. not after each interaction) as these refreshes are costly.
- the other possibility is to precompute the similarity distances between users, items and user-item when it is required and store them in the database. A special class then allows to exploit directly these measures instead of using lots of requests to compute them each time. This last solution is obviously the most scalable as it requires both less requests and less RAM.

These solutions, combined with the previously discussed caching system for recommendations should ensure performance and scalability to the system. It is roughly the method used by main actors in this field such as Amazon. When the dataset becomes huge, recommendations are not computed in real-time but as much pre-processed offline as possible with computations distributed on a grid of machines.

3.6 Deliverables

All sources developed during this project are available for downloading at:

<http://github.com/picalag>

This public git repository includes the Picalag RoR front-end, PServer Java application and an example of ruby-based import script (CityLife scraper)

Documentation and install instructions can be found in appendix B on page 118.

Chapter 4

Evaluation and analysis

4.1 Introduction

In the previous chapters, the context and related work were explained and the solution developed to solve the problem of events aggregation and recommendation was detailed. The system Picalag is now ready to run and ready to be evaluated with real users.

In this chapter, we first present the methodology used to evaluate the system, the hypotheses expected to be confirmed or invalidated and the questions we expect this experiment to answer. In a second part, the results of the experiment will be reported and analysed.

Evaluation is an important step that must be addressed when building a system. Its main goal is to validate that the requirements listed in the previous chapter are effectively solved by the system. The second reason we need a usability study and test with final users is to unveil future improvements by collecting feedbacks and get opinions about whether or not the interface is solving the task.

4.2 Evaluation methodology

For this evaluation, an instance of the Picalag system was set up on a server accessible online through a sub-domain of the school of computer science website. Both the Ruby on Rails front end application and the PServer Java back end were running on a virtual machine (powered by Oracle's VirtualBox and set up with Turnkey Rails

Ubuntu-based Linux distribution) hosted on a desktop computer in the school. The host machine has a powerful i7 CPU but the usage was not dedicated to the virtual machine. This constraint limited the amount of available RAM memory to 1024 Mb. For this reason, the import script used to feed the system was running from another computer and sent the data using the RESTful API remotely.

The system has been fed for several days scraping the CityLife website. This was the unique source for this experiment as this website contains around one thousand events each day. Obviously, Calagator and Picalag are designed to accept lots of different inputs but we considered CityLife as sufficient for this test with a wide variety of different events (e.g. cinema, live music, theatre, days out). Therefore, the imported events were representative of what Manchester can offer and almost any important venues in the city were covered and imported in the system.

The experiment aims to make volunteer users (possibly final users if the system happens to be used in a production environment as a real service) regularly browse the website as if they are looking for some events to attend for a day or a night out for a few minutes per day, over 6 days. Feedbacks are then collected thanks to questionnaires. This study is completely online and no contact with the participants is required from the tester. It also means that the participants can use the system remotely whenever and wherever suit them best. Nevertheless, the downsides of a several days online study are numerous: the participant could be unavailable some days or may forget to use the system or to complete the questionnaires.

At the beginning of the experiment, each participant has to complete a first initial form. This quick questionnaire contains the information about the experiment (e.g. all the ethics related information), requirements to participate as well as a description of the process the participant will go through. It is divided into two parts. The first one is an electronic consent form. This is required for any experiment involving human participants. The second part is designed to collect demographic information about the participant and data about his usage of computers, web, aggregator systems and recommender systems. This initial questionnaire is joined in appendix A on page 107.

The user is then redirected to Picalag homepage to register in the system. Quickstart instructions are given in this page as a “get started” guide: find your favorite venues and mark them to have an easy access to what is going on these venues, browse events that interest you and check out the recommendations we generated for you on your picalag dashboard page. Participants are asked to use the system around five

or ten minutes each days for six days just as a normal case (the scenario is: “you are looking for something to do for a day or a night out”). Except these quick guidelines, no other instructions are given to the user as we want to evaluate if the interface is easy and clear enough to solve the task efficiently.

After each session (~ 5 minutes browsing), the participant is asked to complete a quick form to collect feedback about his experience while using the website. You can see the questionnaire in appendix A, section A.2 on page 112.

When the website is used, several parameters are logged to be analysed for the evaluation. PServer database is also saved everyday for further inspections. Thanks to these information, we can track the evolution of both the system and users’ satisfaction over the period of the experiment.

At the end of the experiment, each participant completes a final form to evaluate the system overall usability and usefulness. This form should give us an idea about the user interface and whether or not it is effectively solving the tasks it was designed for. The final questionnaire also offers the opportunity to the participant to give some comments about the whole experiment. Daily forms also had a “comment” question, but they were mostly aimed at reporting immediate feeling about the session (such as bugs or things that struck the user). A sample of the final questionnaire can be found in appendix A.3 on page 114.

4.3 Evaluations

The evaluation was carried out to validate or invalidate the relevance of the system. More precisely, this experiment is designed to give an idea about how efficient the intelligent part of the system (PServer) and the front end (Picalag web-site) are at solving the problem of event aggregation and recommendation.

This section gives a list of the different points evaluated, the objectives, hypotheses and questions we expect the experiment to achieve, validate or answer.

4.3.1 Evaluation 1: relevance of the intelligent part

With this evaluation, we expect to understand if the recommendations the intelligent part generate are relevant to the user. Moreover, this evaluation should verify if there is any improvement in the relevance of the recommendations pushed to the user from one day to the next one.

Hypotheses

For this part of the evaluation, the main hypothesis is that Picalag is able to extract the relevant events based on user’s interests to recommend them to the user. Picalag should also be able to filter out the events that are not relevant to the user.

As the system is adaptive, we also make the hypothesis the more somebody uses the system, the better the recommendations become. As a consequence, we expect the satisfaction of the user and the precision of the recommendations to increase during the week of the experiment as the system “learns“ users’ interests.

Objectives

This evaluation has two objectives. First, we want to make sure the recommender systems are effectively working as they should. The second objective is to ensure the learnability of the system is working from a user’s point of view.

Questions

With this test we want to answer several questions. The more important one is obviously “is the system able to recommend relevant events?” as it is the goal of this project. However, the collected data should also make us able to answer other questions such as: can the system adapt to capture user’s taste and is the captured profile accurate?

Data

The data used to assess this aspect of the system is mostly collected thanks to the “end of session” questionnaire (see A.2) completed daily by participants after they used the system. This is justified by the fact we want to know participant’s opinions and questionnaire are the easiest and most precise way to collect them.

We also monitored users’ clicks on recommended items’ links. If the data collected is sufficient, we can check if any recommendation module statistically has a different “click behaviour” than random recommendations (i.e. if the recommendations suggested by a module generate statistically more clicks than random recommendations, it means this module is statistically relevant to the user).

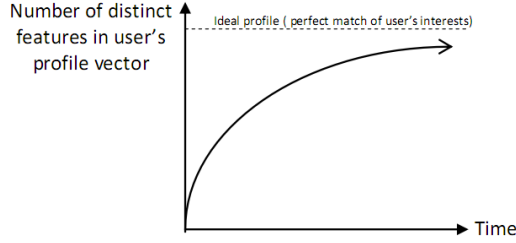


Figure 4.1 – User profile saturation

4.3.2 Evaluation 2: data structure, storage and performance

By analysing data structure and storage as well as some performance indicators, we aim to understand if the system is working and efficient in a more technical point of view. Even if the duration of the experiment is short compared to what a real service should bear, it can give a good idea of the scalability of the system and the evolution of the resources it could require.

Hypotheses

This evaluation is not really driven by hypothesis. It is mostly empirical and aims to see how the system could react if used as a real service.

However, concerning the data structure and storage of users' profiles, we can expect the number of weighted features to saturate with the time. Indeed, as the system learns user's interests, more and more features related to his tastes will be added to user's vector. At one point, we can make the hypothesis the system would have learnt all the features representing concepts of interest for the user and no new words will be added (nevertheless, weights will continue to evolve). This idea is summarized in figure 4.1. After a while, the system should tend to discover the unique vector representing user's interest, even if this description is highly simplistic (it is obvious user's interest cannot be reduced to a single fixed vector).

A similar growth pattern is expected about the total number of features as the total number of events increases (i.e. the system discover more and more concepts but less and less new concepts). This would ensure overlap of event features vectors from a day to an other.

Objectives

The objective of this experiment is to ensure the system can serve the user in a reasonable time, even if the data grows. It should also give an idea of the resources and storage space required by the system as it grows.

Finally, if the hypothesis about user profile saturation is validated, it will be an extra confirmation that the adaptive intelligent part is working correctly and is able to capture user's interest. Evaluation 1 and 2 are then assessing two aspects of the learnability of the system (respectively from user and mathematical point of view).

Questions

This evaluation should bring an answer to questions such as “is the system sustainable?” (i.e. can it continue serving the user effectively when it scales/grows?). We also expect this experiment to complete the answer to some of the questions proposed in evaluation 1.

Data

For this part of the evaluation, we logged the response time of PServer for any request received. We also dumped PServer database every day to track evolution of data structure, storage space required and user profiles.

4.3.3 Evaluation 3: user feedback

Finally, the main evaluation is made by collecting participants' feedback. The goal of this project is to build an application which serves the user and helps him in finding events. This can only be achieved if user is satisfied with the product.

This kind of satisfaction survey can also give an idea about whether or not such a service could be successful if launched in a real production environment. In a project which uses user-centered design like this one, feedbacks also allow to discover ways to improve the product in future development rounds and versions as well as discover bugs. It is therefore very important.

Even if HCI is clearly assessed with this survey, it is not designed as a proper HCI evaluation that could involve heavier processes as eye tracking, “think aloud” method,

video recording and one-to-one structured or informal interview. Instead, we only try to measure satisfaction about the interface “naively”, with questionnaires only (and a few open questions).

Hypothesis

The hypothesis is no less than the one given in the project statement in the introduction of this dissertation. Indeed, this project was based on the idea that user experience could be improved when looking for events by the use of an aggregator and a recommender system.

This survey should be able to validate this hypothesis or not, even if the number of participants does not allow to draw precise conclusions about larger populations.

Objectives

The objective is first to collect impression about this particular system and ideas to make it better in the future if it is considered as useful by the users. More generally, this evaluation is also carried out to understand if this type of system is relevant.

As we it was discussed in the background chapter, calendar events aggregation and (even more) recommendation are a not-so-studied field. Only few surveys can be found in the literature to understand the usefulness of these systems ([53] is the only real size study about events recommendation found during the review of papers for this dissertation).

Questions

The questions it should help to answer are directly derived from the hypothesis and the objectives. For instance: Would people use our service if it was publicly available? Would it replace other sources of information for events?

Data

The data used for this part of the evaluation is only collected thanks to questionnaires completed by participants at the end of each browsing session (see A.2) and at the very end of the experiment (see A.3).

4.4 Results

This section is devoted to evaluation results. They will be analysed and discussed to challenge the hypotheses we made in the previous section.

4.4.1 Participants sample

As the ethics approval took some time to be obtained for this experiment, the participants were recruited thanks to social networks on a group page of the University of Manchester. Indeed, at this period of the year it appeared difficult to find volunteers quickly using other means of communication.

For this reason, the participants are mostly students: out of 12 participants, 10 were postgraduate students doing an MSc or a PHD in the university. Only one user was a professional (research technician) and one was about to start a Master degree after one gap year working in Manchester. Although they are in majority part of the school of computer science, two of them happened to be studying languages. The average age of the participants is 26.5 years old (age range from 22 to 30) and 58% were males (7).

Even if this sample is not particularly representative of the population living in Manchester, we have to take into account that this city is lively thanks to its huge campuses and student communities. It means that, as far as nights out are concerned, events are influenced by student life and students are the main targets for venues. Students are also in our opinion the population who could more likely be interested in a service as Picalag in real life case.

Concerning their web habits, it is interesting to note that although they all considered themselves as computer literate and the majority of them (67%) declared they usually spend more than three hours on the web each day, half of them did not know what an aggregator is before they were given the Wikipedia definition and only 32% were sometimes using an aggregation application (for news content). Half of them regularly use an application to manage their time (e.g. Google calendar or their smartphone).

When looking for an event to attend, the web is a source of both inspiration for new events or venues and information about what happens in their favorite venues for 75% of the sample. Moreover, 67% of them go directly on their favorite venues websites to know what is going on. They also are 67% to use Facebook. Only 34% use other sources such as CityLife.

Recommender systems are far better known than aggregators as 83% use them frequently, mostly to broaden their interest and sometimes get surprised thanks to the generated suggestions (67%) but with high precision expectation (50% want the recommender system to strictly stick to their interest and 33% believe it should be unfailingly accurate). Half of the sample agree that RSs are a good way to save time. When asked what kind of recommendations they trust, 64% said they enjoyed the Amazon-style “people who purchased this item also purchased...” suggestions and 55% like to have a list of other similar items recommended when browsing a particular item. Other users are usually relatively trusted thanks to recommendations as “most popular items” and “similar users viewed...” (respectively 45% and 55%). Personal profile matching looks more suspicious as only 36% of the sample trust this method.

4.4.2 Data feed

During this experiment, only events present on CityLife website were extracted and imported in the system thanks to an automated import script developed in Ruby. The script was run manually as it was set up on a personal laptop which is not running 24-7. Events were imported in advanced (usually up to two days in advance), but the script was run only every two days as CityLife last minute updates are not frequent.

Events were scraped from the 21st to the 30th of August 2011, as shown in table 4.1. The last column of the table shows the evolution of the total number of events in PServer database at a given date (obtained thanks to daily data dumping from 22nd of August).

Date	Number of events for this day	Total events in PServer db
21/08/2011	822	3,341
22/08/2011	1,192	5,935
23/08/2011	1,304	5,935
24/08/2011	1,269	7,051
25/08/2011	1,348	7,051
26/08/2011	1,327	9,338
27/08/2011	1,179	9,338
28/08/2011	948	11,894
29/08/2011	1,196	11,894
30/08/2011	1,309	11,894

Table 4.1 – Events importation

4.4.3 Data structure, storage and performance

Response time of Content-based recommender system

The parameter studied is the average response time when a request is received by PServer concerning recommendation of similar events based on content (on event page) and events matching user's profile (on my picalag dashboard). Note that the time logged corresponds to the processing time between the reception of the request and the moment the response is fired (on PServer side only, so we do not include communications from front-end to back-end). This time is computed by the Active-JDBCFilter servlet filter described in the previous chapter.

What we want to know here is if the number of events on the database has an impact on the response time of the system. This can give us a clue about the scalability of the system and its sustainability as the database grows. Of course, the response time in itself is highly dependant on the configuration of the host machine. As a consequence, it is not really relevant to know that on this instance of the system the content-based module responded with an average of 2.7 seconds to the users' requests (even if it is not that bad given that the recommendations were computed live).

What is important to notice here, is that as recommendations require several seconds to be generated, the choice of AJAX to make it asynchronous is adapted. As other recommendations types (e.g. random or popularity based) are much quicker to compute, the user certainly does not want to wait for that long all recommendations are generated to see the page loaded. This allows to print the recommendations as they arrived (user can start looking at the popular events before recommendations based on his profile are ready).

The second important information we can extract from graph presented in figure 4.2, is that data series representing the total number of events in PServer database and the data series representing response time for either CB filtering similar to an event or similar to a user profile are not correlated. Indeed, the correlation coefficient is $r^2=0.1888$ in the former case and $r^2=0.1552$ in the latter.

As a consequence, we can be pretty confident in the scalability of this module. This is due to the fact we limited the recommendations to events taking place in a given day only (same day as an event or day chosen by the user). Then, the number of similarity distances to compute and compare depends on the number of events taking place on this day. Nevertheless, to improve performances, pre-computation and caching of

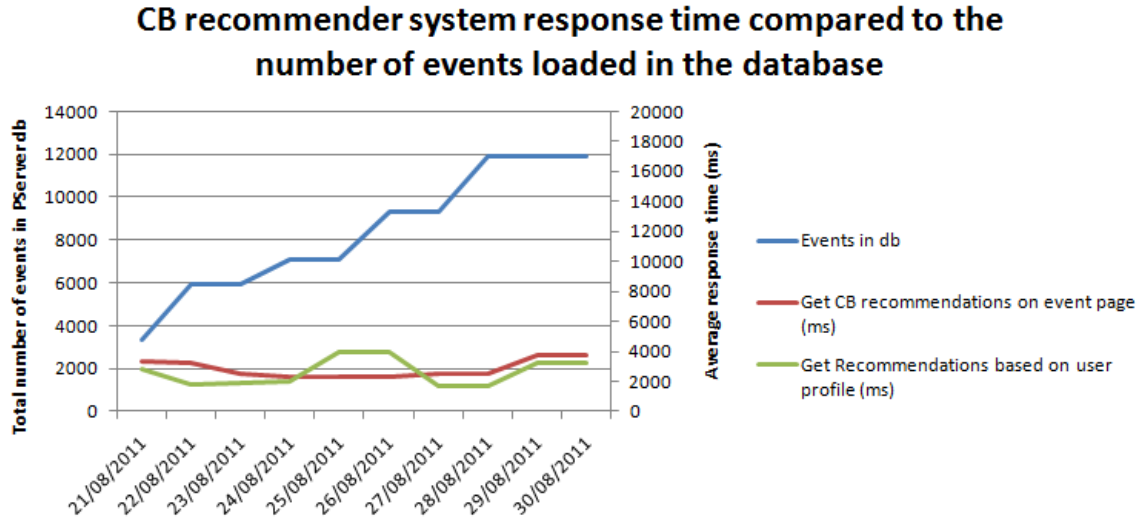


Figure 4.2 – Content-based RS response time

the recommendations can be done. Database access could also be optimised on this recommender system.

Saturation of the total number of features and of user profile

Unfortunately, the experiment was not carried out on a long enough period to be able to validate these hypotheses. The two charts in figure 4.3 show the results obtained during the evaluation by analysing the database exported files.

Saturation hypothesis seems logical but our data cannot confirm it and no saturation time can be obtained from it. It is probable that over a longer period, several weeks, the saturation tendency would be more clearly visible.

Data storage

After 10 days, PServer database size is around 33 Mb with only few users registered and a low traffic and usage. Chart 4.4 shows the evolution of the size of each table in the database. As we can see, the database tends to grow relatively linearly of 2.5 Mb per day. Unsurprisingly, the biggest tables are events_weighted_features and events_boolean_features which store events vectors. These two tables represent nearly 95% of the total database size. For its part, Picalag front end database which stores the events and venues to be displayed to the user has a similar growth pattern with a final size of 27.8 Mb at the end of the experiment.

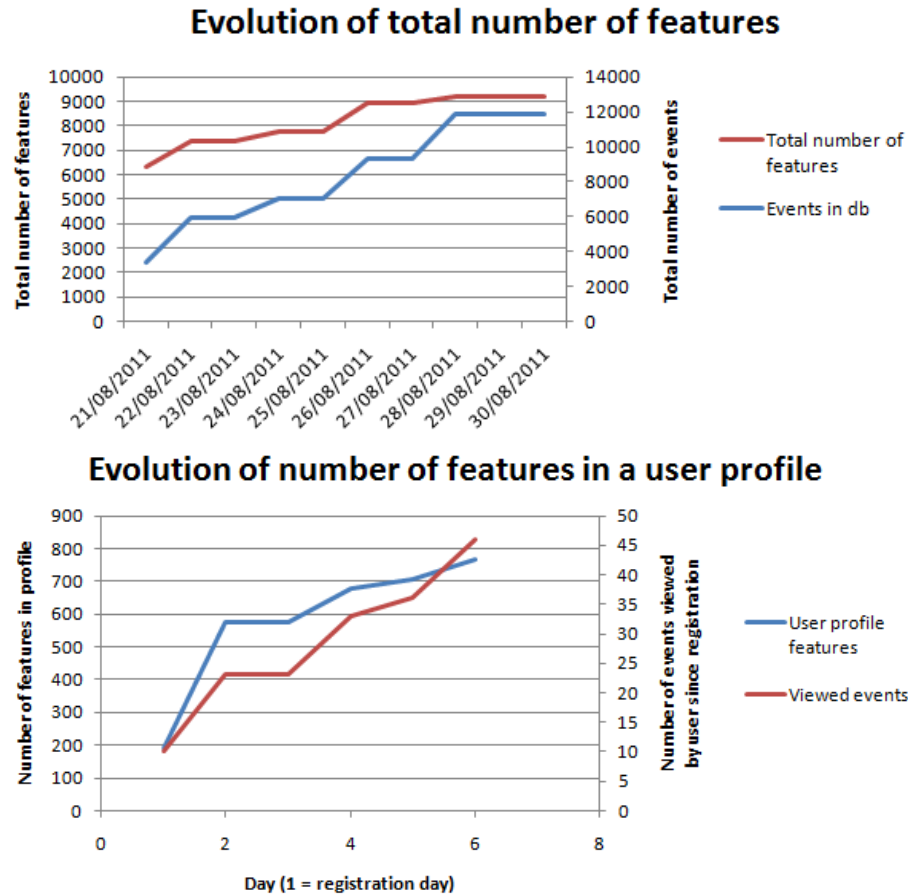


Figure 4.3 – Evolution of number of features in PServer and in a user profile

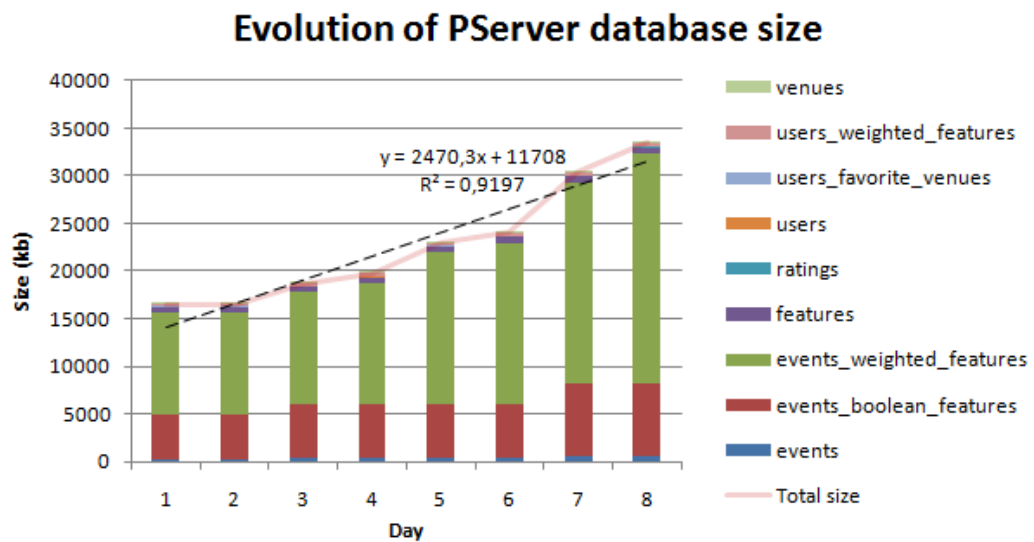


Figure 4.4 – Evolution of PServer Database size

Nevertheless, even if the growth of database is quick, it is not really a problem for scalability and long term use of the system. Indeed, as we discussed in the background chapter when we were wondering which strategy suited best for our problem (see 2.2.3 on page 30), the data we are dealing with has a very short life time (i.e. it is perishable). Once the event took place, we do not really need to keep it in database any more as only a negligible number of users will look for information about a past event. Moreover, as we bound our recommendations to one given day and it is again unlikely the user will look for suggestions of past events, databases could be cleaned of all the past and useless event vectors and descriptions. This is possible as the system maintains a user profile vector which is updated each time the user interacts with an event. Thus, we do not need to keep past event vector in the database as the learning part is in user profile vector.

It is then recommended in a future version of the system to add a script which would regularly clean both databases from past events (for instance all events older than 2 weeks) to keep a database with an acceptable size without any disadvantage for the user. However, user ratings should be kept as they are used to determine similarity and user neighbourhood for collaborative filtering.

4.4.4 Relevance of the intelligent part and user feedbacks

Discussion about the quality and amount of data collected

The experiment took place at the end of August which is busy for most of the students and the participants did not really have time to spend on it. Everybody had more important things to do such as dissertations, or journeys to look for a flat or attend job interviews for next year. Participants were absolutely free to stop the experiment whenever they decided and that is what they did as urgent duties were first in their to do list.

For this reason, none of the actually registered participants used the system every day even if the experiment was not particularly time demanding. As a consequence, the data collected is not as accurate as expected and the experiment was not exactly executed as it was meant to (and described in above sections). For instance, it is not possible to really track the evolution of participants' satisfaction over the 6 days as many of them only completed the form for the two first days. In the following paragraphs, when an evolution was supposed to be found over 6 days, we will present graphs for the 2 first days and the total result for all the end of sessions form

completed by participants as there is not enough extra data to reasonably do some statistical analysis.

This section presents data we could collect but unfortunately the quality and amount of data collected does not reach the expectations for the designed evaluation. The project time frame did not allow to carry an extra experiment out or redesign it completely to adapt it better to the participants constraints. As discussed in 4.2, we initially wanted the users to feel in a real use case: in front of their computer looking for events to attend for the current day (and the website was up-to-date with real events of the day). It probably would have been easier and more adapted to do the experiment in one day by simulating working days on the system with pre-loaded events from the previous week (as [52] did for the evaluation of his MSc “intelligent RSS aggregator” project).

Recommendation modules

During the experiment, the system recommended 2184 events and 147 venues to the users. We recorded clicks on links to the suggested items for each module. The results are presented in table 4.2.

The most popular recommendation modules are “similar events” recommendations printed when browsing an event and recommendations matching user profile on My Picalag dashboard. These two modules have the best click rate. Surprisingly, few users followed the links of recommendations appearing in “most popular events” module. This click behaviour is different from data collected in [53] and even in contradiction with the participants ranking of the modules (see figure 4.5). It is probable that results would have been different with a longer study and more participants.

We also see that pure collaborative filtering modules suffer from cold start problem as few ratings are available in this experiment. This explains why the “events based on similar users ratings” was not able to recommend a single event. Similarly, the venue collaborative recommender only suggested 4 venues. These modules have to be optimised for the problem of event recommendation (see previous chapter) and require more users and more browsing interactions (ratings) to work properly.

Recommendations

The quality of recommendations have been appreciated by users as it is shown in charts in figures 4.6, 4.7, 4.8, 4.9 and 4.10.

Module	Recommendations	Clicks	% clicked
CB event	735	36	4.90
CB user	386	12	3.11
CF event	302	3	0.99
CF user	0	0	0
Popular events	346	8	2.01
Random events	415	8	1.93
Popular venues	143	4	2.80
CF venues	4	0	0

CB event: Similar events based on content (event page)

CB user: Events recommended based on user profile (My Picalag dashboard)

CF event: “People who viewed this event also viewed...” (event page)

CF user: Events recommended to user based on what other people liked (ratings) (My Picalag dashboard)

CF venue: Venues recommended based on other people’s favorite venues and Mahout

CF algorithm (favorite venues page)

Table 4.2 – Recommendations during the experiment

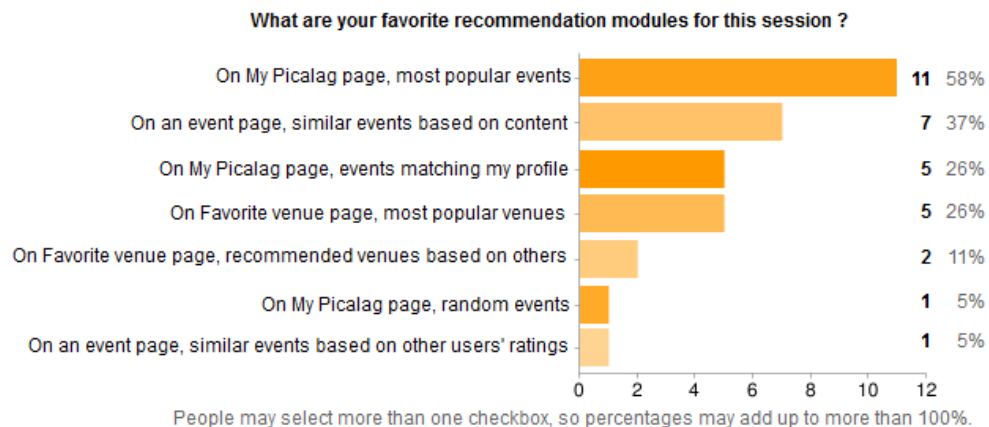


Figure 4.5 – Recommendation modules ranking

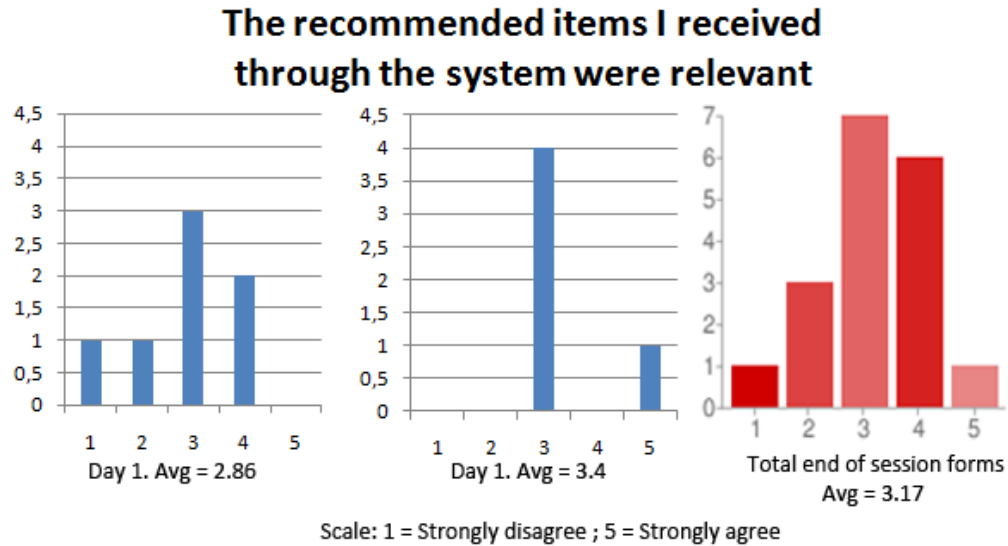


Figure 4.6 – Relevance of recommendations

As it was expected, first day suggestions were not really accurate and did not match users' taste very precisely. However, by day 2 participants reported a better relevance to their interest. This tendency is confirmed in chart 4.9. Even if the experiment data is probably too limited to be able to conclude the system works perfectly, it confirms the intelligent part does its job properly. Therefore, the learnability of the system is visible and validated.

When asked to grade the overall quality of recommendations they received from the system, participants awarded nearly 7/10 (see figure 4.10). This grade would not have been this high if the captured interests were not pretty accurate. Moreover, the users declared they could understand why recommendations were made given the previous events they browsed (figure 4.8). This allows us to be confident in the trust the user can have in the system (and trust is important when dealing with RSs).

System and interface

The trust of user in the system is confirmed in chart 4.12. This probably comes from both the fact the users felt in control (see figure 4.13) and comfortable (see figure 4.11) during the use of the system and the explanations given for each recommendation (e.g. grade, % profile matching) as discussed in previous chapter about implementation of Picalag.

Participants liked the interface as they said it is "simple" and "contains lots of in-

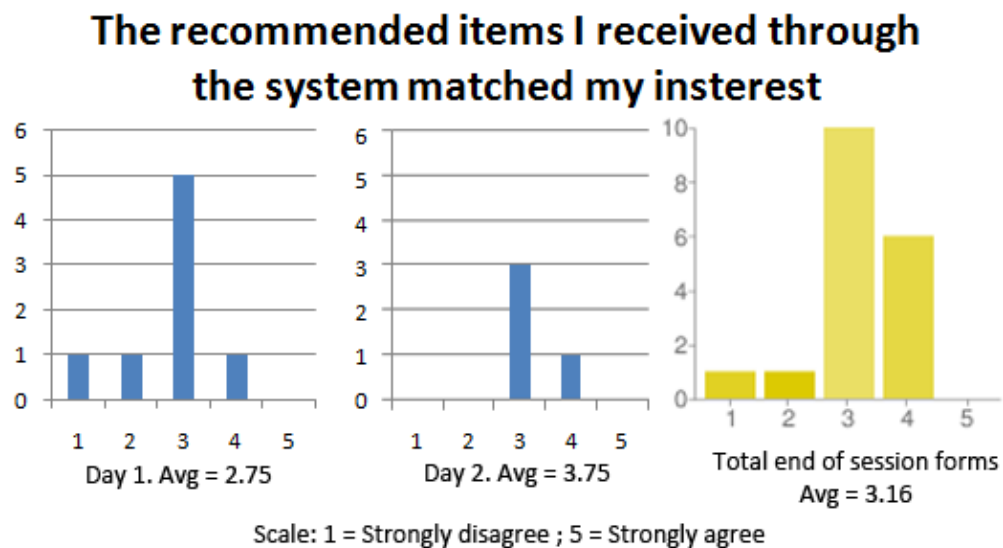


Figure 4.7 – Match recommendations - user interests

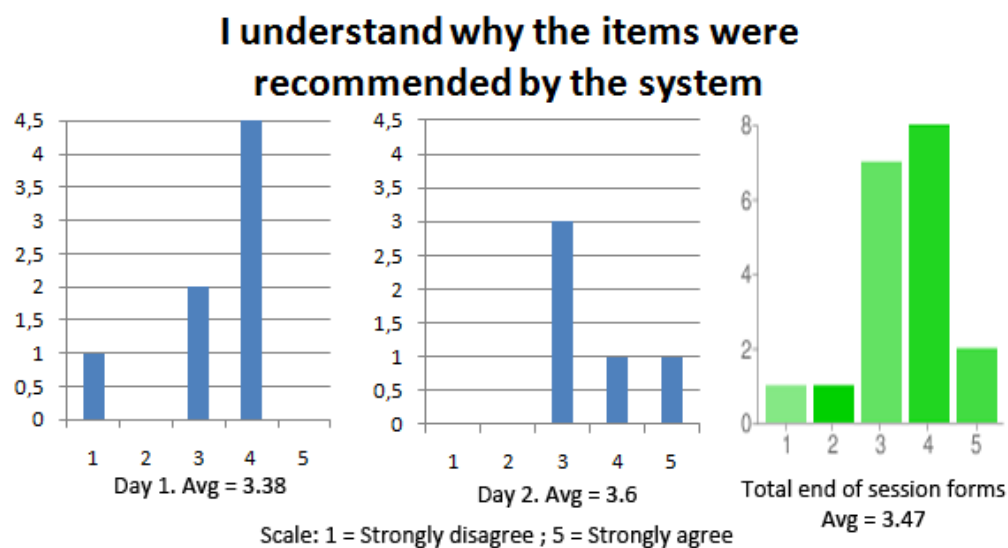


Figure 4.8 – Coherence of the recommendations

**Did you feel the quality of recommendations was improving
during the week as the system learnt your interests ?**

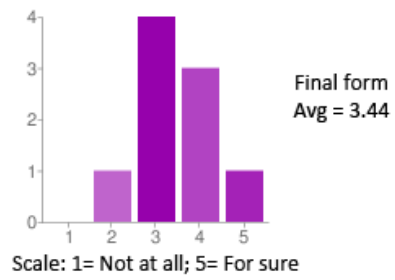


Figure 4.9 – Evolution of the quality of the recommendations

Can you give a grade to the system for the quality of recommendations you received ?

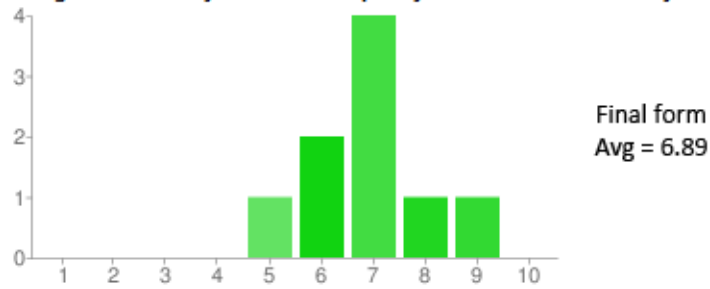


Figure 4.10 – Recommendations grade

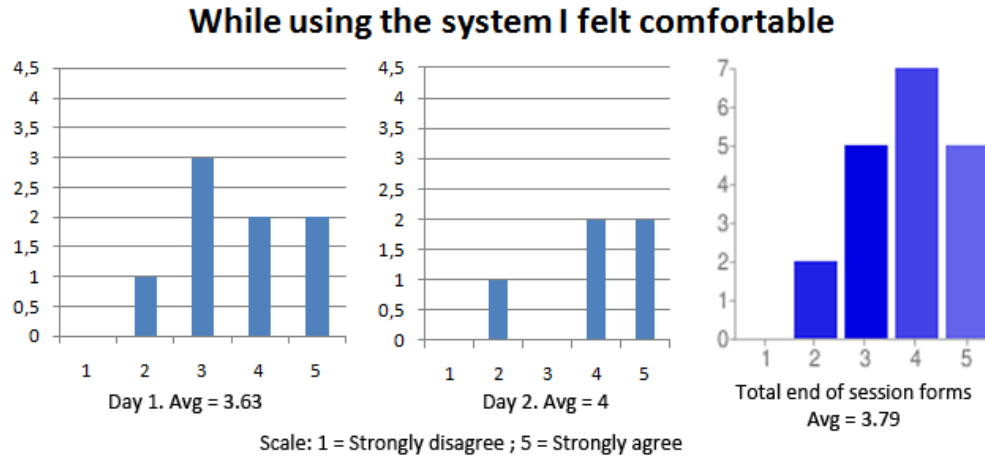


Figure 4.11 – Is the interface comfortable?

formation”. The clarity of the website is mostly the result of Calagator development team as the extra features were most of the time integrated on the pre-existent design. Even if some people pointed out the fact it is not very attractive, a design which can be both simple and show a lot of information is rare enough to be noticed: only 11% of the users declared the interface was not effective for solving the task at a point of the experiment (see chart 4.14), which is a good score.

In charts contained in figures 4.16 and 4.17, users confirm the system is useful and novel. The idea of grouping sources in a unique website for events seems to be a service which can make people save time and which could be famous if it was proposed as a real service (89% of the participants said they would probably use it).

Nevertheless, before it can be used as a real service it has to be improved. The main problem about the system as it is currently implemented is its responsiveness: a majority of participant said the system was too slow to be really used as it is now. Time optimisation will have to be the main concern for future developments before the system can run in a production environment with more users.

System Usability Scale and User Satisfaction

To evaluate system usability, we used the well-known System Usability Scale (SUS). It is a widely used evaluation method based on 10 questions. Introduced in 1986 by John Brooke, it quickly became a standard in the industry as it is quick and known as accurate.

This scale was designed to evaluate the ease of use of the system only but [43] recently

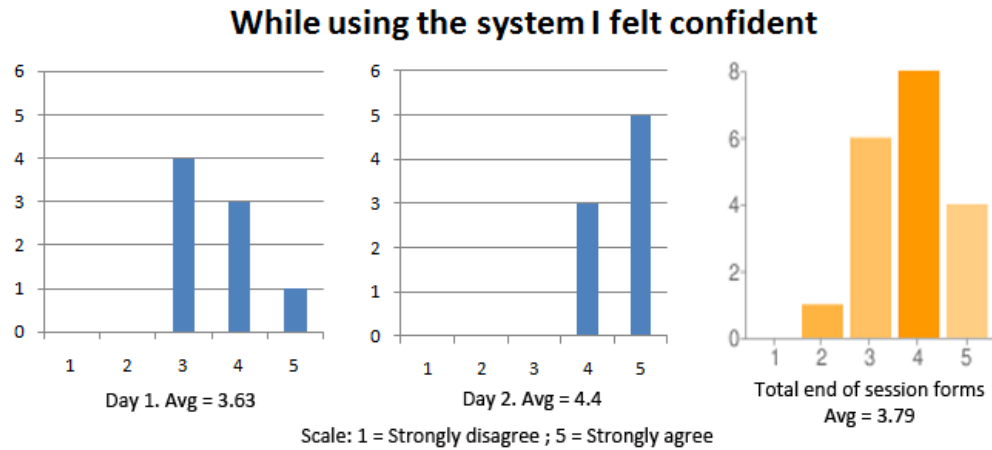


Figure 4.12 – Trust in the system

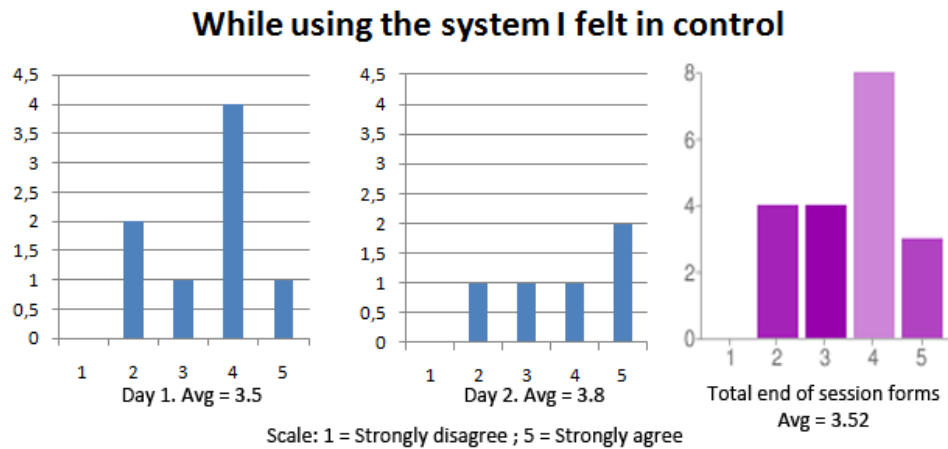


Figure 4.13 – Control over the system

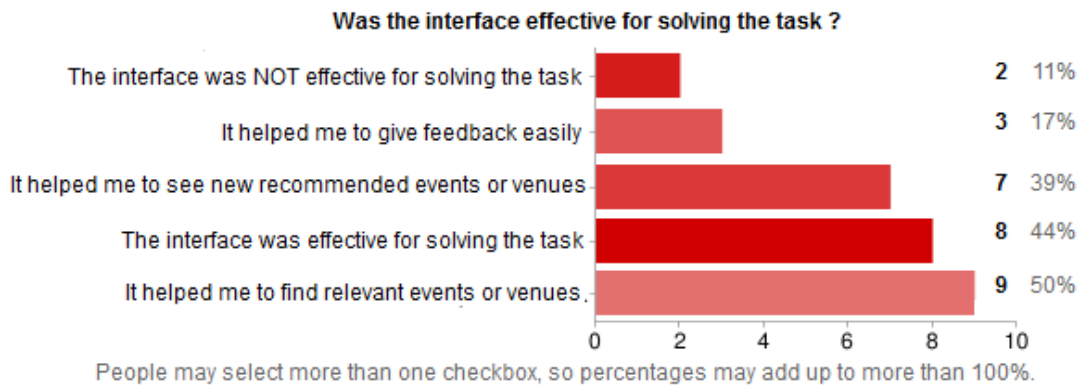


Figure 4.14 – Quality of the interface

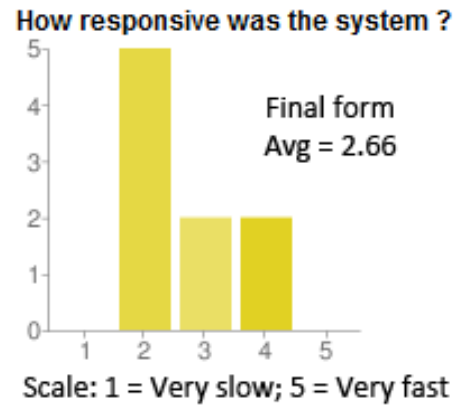


Figure 4.15 – Responsiveness of the system

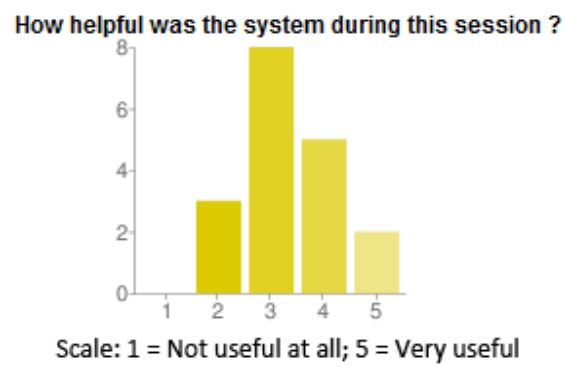
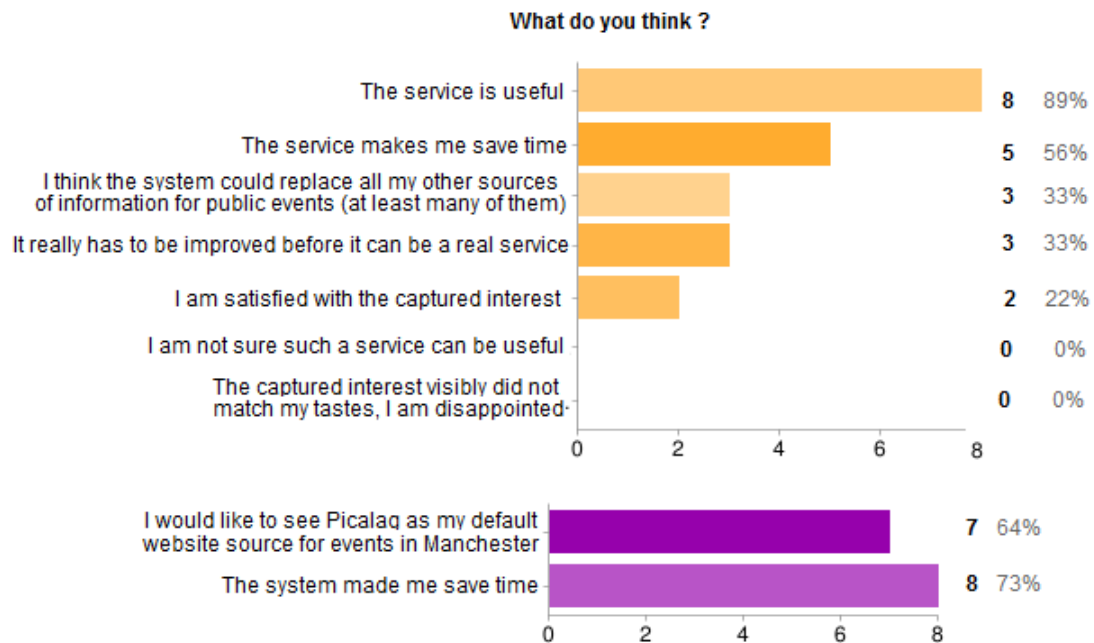
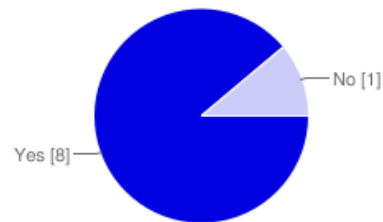


Figure 4.16 – Usefulness of the system



People may select more than one checkbox, so percentages may add up to more than 100%.

If this system comes out as a real service would you use it ?



Can you give a grade from 1 to 10 for the system during this session ?

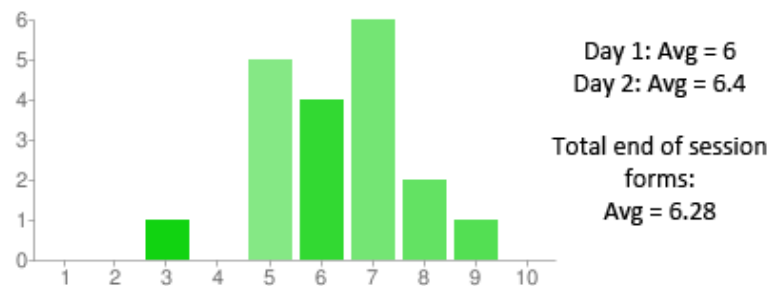


Figure 4.17 – Users opinions

demonstrated it has more than one dimension. In other words, it can also be used to measure satisfaction, learnability and usability. Thus, it is relevant to use SUS to validate our main hypothesis.

To that end, we simply added the ten questions of SUS in our final questionnaire (see A.3 on page 114). The user has to say on a Likert scale (1 to 5) if he agrees or disagrees the following statements [8]:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

Results obtained are reported in figure 4.18 on the next page. Then, we subtract 1 to user response on every odd statements (left charts column) and we subtract the response from 5 for even statements (right column) to have scores between 0 and 4. For each user, we sum the scores and multiply the result by 2.5 to have a final number between 0 and 100. The average final score is the SUS result. However, even if it is ranged from 0 to 100, it should not be considered as a percentage. Indeed, to interpret the result, the score has to be reported on the chart 4.19.

According to the collected questionnaires, Picalag application gets an average SUS score of 77.78. If we look at the graph coming from [70], we see that this score corresponds to percentile rank of an application in the “top 20%”. To be in the top 10% we should have reached a score of 80.3 and any application with a score lower than 68 is below the average (and below 51 means your application is in the bottom 15%). According to Jeff Sauro, this score corresponds to a B or a B+ on a letter-grades scale from A+ to F.

Therefore, this result is pretty good and as it is correlated with user satisfaction, the main hypothesis of this project is validated. This study confirms the use of

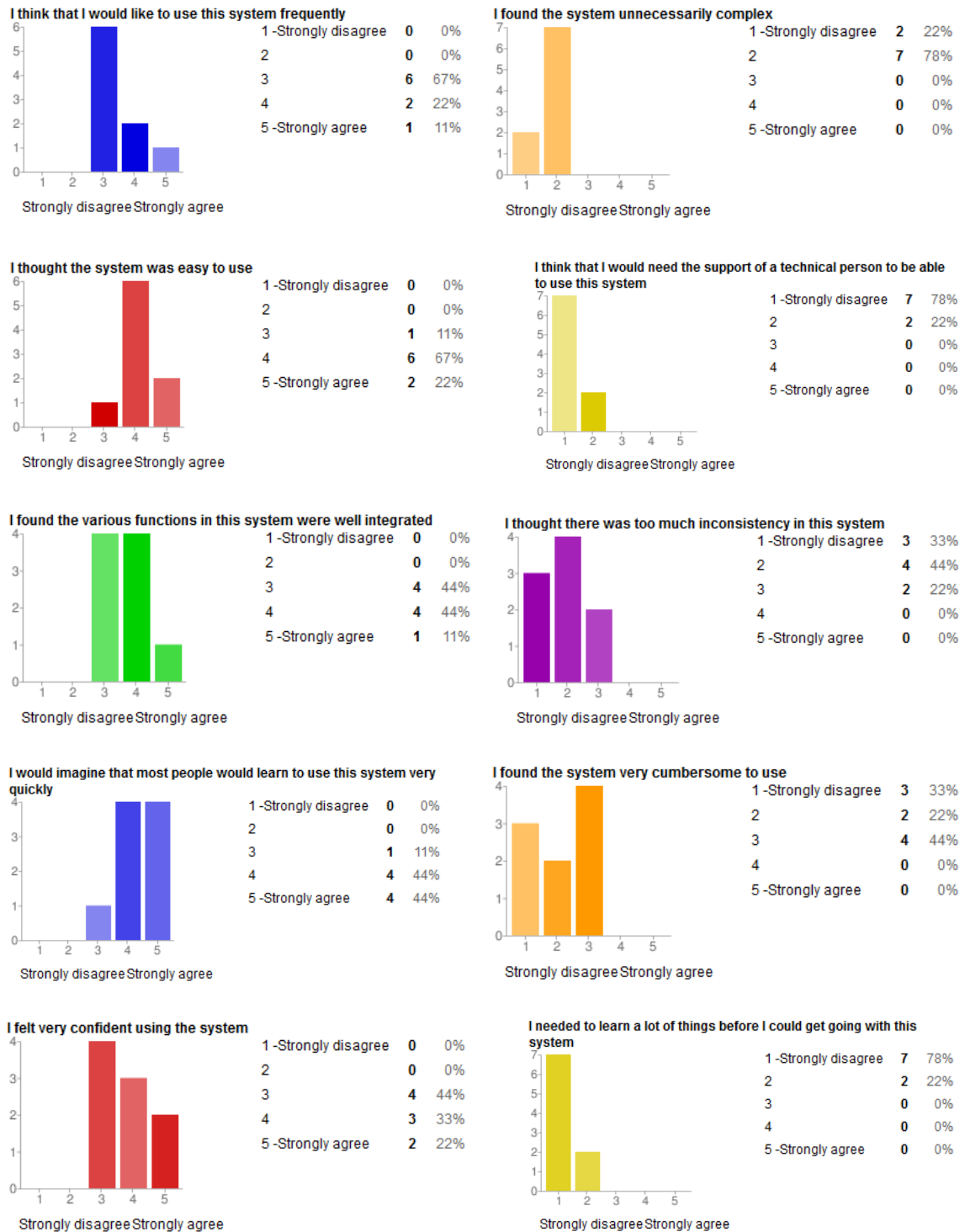


Figure 4.18 – System Usability Scale

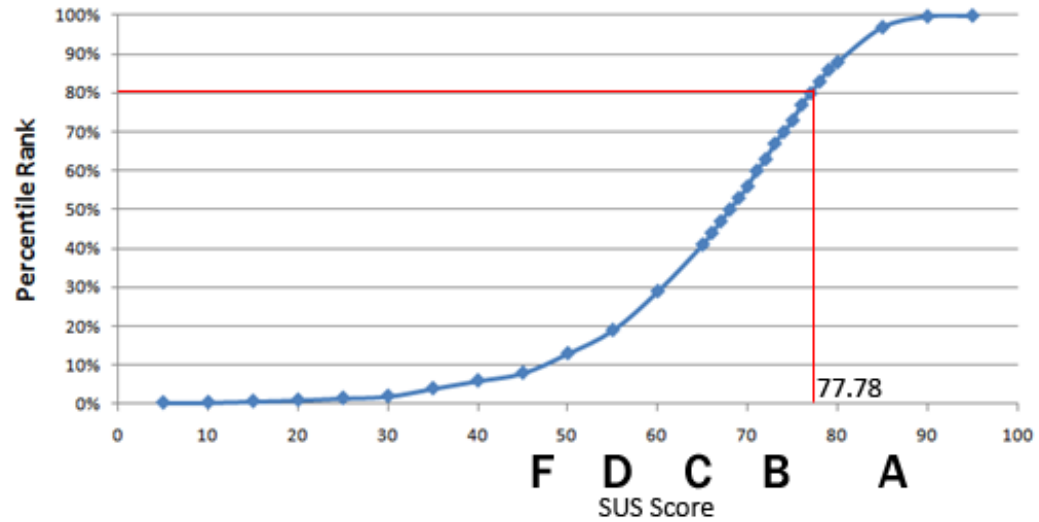


Figure 4.19 – SUS score interpretation [70]

aggregation and recommender systems in the context of calendar events in a system as Picalag improves user's experience. Moreover, we see in charts 4.17 that at least 33% of the tested users consider Picalag could replace some of the sources they are currently browsing to look for events to attend. This means the developed system has a real value added compared with a classic events repository such as CityLife or Upcoming.

Chapter 5

Conclusion

5.1 Summary

We developed a web-based intelligent calendar aggregator to enable users to use a unique source of information for all events in Manchester and enjoy recommendations matching their interest. Picalag is a very pragmatic project with direct possible practical applications. For this reason, the topic proposal as an MSc dissertation project was motivated by a potential end-user need: Mancunians have too many choices offered when they want to choose an event to attend. Nevertheless, by simply aggregating various sources, a new problem occurs known as “information overload” that could make the system difficult or impossible to use. We then made the hypothesis user experience could be dramatically improved with the help of an artificial intelligent agent to pre-filter items on his behalf (i.e. a recommender system).

The aim of the project was to investigate a possible solution and to bring a technical implementation meeting this need. This was achieved by using the classic project organisation in computer science, in 4 steps: make user’s needs explicit by analysis, design a system to fulfil his expectations, implement a prototype, evaluate if the system really addresses the expressed problem and finalise the product.

First of all, a long phase of background research was undertaken to understand the context and the technologies that could be used to solve the tasks. The review of this reading phase and early analysis of the problem are written down in chapter 2. It gives a clear idea about the points we should be careful about and a good insight on available solutions that could be applied to set a recommendation website up.

The second step was the analysis of users’ specific needs and functional requirements

that were used as base for the implementation of the prototype website Picalag. The choice of relevant and coherent technologies as well as the top level architecture were directly derived from these requirements. The system is composed of two main parts: a front end web application and an independent personalisation server. The former is powered by Ruby on Rails framework (this was imposed by the fact we used the existing project Calagator as groundwork) while the latter is developed with Java web-server standards. Each part has its own database allowing the system to run on different host machines and are cross-platform. The system was implemented in a ten weeks time frame using these technologies. Chapter 3 is devoted to explanations and details about this step.

Finally, an evaluation with human participants was carried out to test the system at the end of the project. Between 14 and 8 participants used the system (depending on the days) and feedbacks were collected thanks to the questionnaires attached in appendix A. The initial idea was to track user satisfaction and feedbacks over 6 days but various circumstances prevented us from collecting the data as planned. Due to time limitation, a new evaluation could not be tried again. Nevertheless, the collected data was still interesting enough to be used for basic statistical analysis. The experiment design and results collected are described and discussed in chapter 4 dedicated to this evaluation phase. The aim of this study was to find out the subjective opinion of users about the system, its interface, its functionalities, the recommendations, and the concept. The questionnaires also allowed us to access a measure of user satisfaction and validate our main hypothesis about the benefits of a system such as Picalag.

The results of the experiment were very encouraging as many users said they would like to use a similar system if it was offered as a real service on a website and thought it could replace some of the sources they currently use to plan their days and nights out. Although this result should be confirmed with a proper HCI evaluation involving eye tracking, interviews and task completion studies the users were satisfied about the interface and the integration of the functions as they said it is simple and practical. The majority of them gave a positive feedback about Picalag prototype. However, this evaluation also unveiled some necessary improvements to address before the system could actually be used as a public service. The following section will give an idea of some enhancements for future work.

5.2 Future work

Picalag was meant to evolve and be enhanced. To that end, it has been released under open-source license on a public git repository (see 3.6 for details). The prototype developed for this project can be improved in many directions as the time frame allocated for MSc Dissertation was too limited to make the system perfect.

Systems synchronisation

As it was described in a previous chapter, the system is composed of two main independent parts. Each of them can run on a different machine as far as they are connected together and can reach each other for http requests/responses. Databases then should be synchronised (thus the systems should be synchronised).

For the moment, the communication protocol we adopted is a RESTful API. However, a big disadvantage of using HTTP requests to keep the systems updated is that if the back-end application is not running when the requests arrive or if the request is lost the two servers get de-synchronized. In the case of Picalag, it was not a real problem as the front-end was designed to be still running and efficient if deprived of its back-end: events and venues could still be browsed, but user interactions were not recorded and no recommendations were made. If a new event is added to the system while PServer is down, this event will not be recommended and this situation only affects this event (thanks to the high perishability of this kind of item it is not a big problem).

The only critical point we identified and which could have a huge impact for user would be the case of a registration of a new user account while PServer is down. This would result in a user that could only receive non-personalised recommendations (most popular events and venues, random events and similar events on event page) and could not manage his favorite venues list permanently.

Synchronisation of systems is a classic yet tough challenge. Many solutions exist as specific protocols or the use of buffer files written on a permanent storage medium (e.g. hard drive if the two parts of the system are on the same host machine).

Response time

As it was reported by users, response time is the main challenge that prevent Picalag from being used as a service right now. The system tends to become slow even with

a low usage. To improve responsiveness of the system, we already discussed several possible ways in the implementation chapter (see 3).

The most promising and probably also the simplest solution would be to pre-compute some of the recommendations offline and store them in the database. For instance, similarity distance between items could be updated in the background to be reused by RS algorithms. This implementation would ensure the system can scale easily and would have the advantage to hide the required computation time from the user. Pre-processing could also be distributed for better efficiency.

Installation of the different parts of the system on dedicated host machines with adapted resources would be a simple other way to reduce response time. However, this could increase the communication problems between the different sub-systems. It is then probable one should solve the synchronisation issue described on the previous paragraph before implementing an instance of the program running on different hosts.

Test Dataset and Collaborative filtering algorithms

In the current version of Picalag, we were not able to create large scale datasets to test the machine learning algorithms. When dealing with information retrieval, datasets that can be found on the internet are not always relevant to test a specific algorithm. Indeed, each problem is different and the algorithms are very often optimised to run over one single type of dataset.

To gather a decent test sample, it will require to collect several hundreds of thousands events from various sources and generate plausible user interaction automatically. The collection would have to cover a good range of events and sources to be relevant. The algorithms could then be tested against this dataset and a real optimisation work could be done.

This is mostly important for Apache Mahout algorithms as we hardly adapted them for our problem in this version of Picalag (and we saw in the evaluation chapter they could not provide many recommendations because of that). Mahout library offers several classes and metrics to evaluate the precision and test the quality of recommendations made by an algorithm over a dataset.

UI design

Even if the current design (which integrates Picalag functions in Calagator original UI) was recognised to be simple, clear and efficient, some users seems to be convinced

the layout should be improved before the system could be used as a real service. The website should then be made more attractive.

Import scripts

So far, only one automated import script has been developed. It scrapes events from the website CityLife. Calagator (thus Picalag) supports many different importation formats and it will then be easy to write automated scripts to push events in these formats to the API.

Having more sources will bring new problems such as the detection of duplicated events and venues in the database to merge them. This is essential for the intelligent part to do its job and nobody wants to see events doubled or tripled on the website. Distance calculation when a new event is added could be a solution (and it would re-use some of the developed tools). For venues, PServer will have to be adapted as it does not store any information about these items. As far as it was possible, this kind of evolution was anticipated when designing PServer and its database.

Security

Security was absolutely not our main concern in this project. The system should then be modified to resist evil-minded actions. For instance, the API are not protected against external calls (an API key module should be added, or some firewall and filtering rules could be written).

Recurrent events

One problem that we did not think about when designing the system is caused by recurrent events. Indeed, if the user liked a dance course event and if this event is present every day in the system, it will be recommended every day to the user (with a high match) based on his profile. This is not acceptable as we could end with a top 10 recommendations that are not relevant recurrent events and the user should mark them as “disliked” to get rid of them (degrading the quality of his profile). A solution could be similar as the one sketched above for duplicates detection.

And more...

Because it is released as an open-source licensed software, Picalag may be re-used and extended as a component in other projects. Several systems could be interested in doing so: the university to promote its seminars and conferences, CityLife which already represents a huge source of events in Manchester which could benefit from it and other modestly emerging events websites such as GoSeeThis [29]. Picalag may also be pushed to initial Calagator's authors and merged to this project.

Finally, further extensions can include mobile interface design to take advantage of the new exciting possibilities offered by now popular smartphones and tablets such as GPS location. Another possibility could be to create social network applications to make the most of social knowledge.

Bibliography

- [1] Academic and Northwestern University Research Technologies. Morphadorner. <http://morphadorner.northwestern.edu/> (visited on 10-08-2011).
- [2] ActiveJDBC. Implementation of active record pattern in java. inspired by ruby on rails activerecord. <http://code.google.com/p/activejdbc/> (visited on 10-08-2011).
- [3] AddToAny. The world's best sharing platform. <http://www.addtoany.com/> (visited on 14-08-2011).
- [4] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17:734–749, June 2005.
- [5] AuthLogic. A simple model based ruby authentication solution. <https://github.com/binarylogic/authlogic> (visited on 11-08-2011).
- [6] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 46–54, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
- [7] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. pages 43–52. Morgan Kaufmann, 1998.
- [8] J. Brooke. Sus: A quick and dirty usability scale. In P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, editors, *Usability evaluation in industry*. Taylor and Francis, London, 1996.
- [9] Robin Burke. Hybrid web recommender systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 377–408. Springer Berlin / Heidelberg, 2007.

- [10] Robin Burke and Maryam Ramezani. *Recommender Systems Handbook*, chapter Chapter 11: Matching Recommendation Technologies and Domains, pages 367–386. Springer, 2011.
- [11] J Lu G Zhang C Cornelis, X Guo. A fuzzy relational approach to event recommendation. In *Proceedings of 2nd Indian International Conference on Artificial Intelligence (IICAI'05)*, pages 2231–2243. Indian International Conference on Artificial Intelligence, 2005.
- [12] Calagator. Portland’s tech calendar. <http://calagator.org/> (visited on 09-05-2011).
- [13] F Carmagnola, F Cena, L Console, and O Cortassa. icity: an adaptive social mobile guide for cultural events. 2006.
- [14] John M. Carroll and Mary Beth Rosson. *Paradox of the active user*, pages 80–111. MIT Press, Cambridge, MA, USA, 1987.
- [15] CityLife. Your definite guide to what’s on! <http://www.citylife.co.uk/> (visited on 09-05-2011).
- [16] Gordana Crnkovic. Constructive research and info-computational knowledge generation. In Lorenzo Magnani, Walter Carnielli, and Claudio Pizzi, editors, *Model-Based Reasoning in Science and Technology*, volume 314 of *Studies in Computational Intelligence*, pages 359–380. Springer Berlin / Heidelberg, 2010.
- [17] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web, WWW '07*, pages 271–280, New York, NY, USA, 2007. ACM.
- [18] DBpedia. Wikipedia knowledge base. <http://dbpedia.org/> (visited on 08-05-2011).
- [19] Toon De Pessemier, Sam Coppens, Kristof Geebelen, Chris Vleugels, Stijn Bannier, Erik Mannens, Kris Vanhecke, and Luc Martens. Collaborative recommendations with content-based filters for cultural activities via a scalable event distribution platform. *Multimedia Tools and Applications*, pages 1–47, 2011. 10.1007/s11042-010-0715-8.
- [20] Christian Desrosiers and George Karypis. *Recommender Systems Handbook*, chapter Chapter 4: A Comprehensive Survey of Neighborhood-based Recommendation Methods, pages 107–144. Springer, 2011.
- [21] Shelley Doll. Agile programming works for the solo developer, August 2007.

- [22] ElmCity. The elmcity calendar curation project. <http://elmcity.cloudapp.net/> (visited on 10-08-2011).
- [23] Lior Rokach Francesco Ricci and Bracha Shapira. *Recommender Systems Handbook*, chapter Chapter 1: Introduction, pages 1–38. Springer, 2011.
- [24] FuseCal. Fusecal project blog. <http://blog.fusecal.com/> (visited on 10-08-2011).
- [25] GeoNames. Geographical database. <http://www.geonames.org/> (visited on 08-05-2011).
- [26] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35:61–70, December 1992.
- [27] Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, pages 439–446, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.
- [28] GoogleCode. Calagator google code page. <http://code.google.com/p/calagator/> (visited on 09-05-2011).
- [29] GoSeeThis. Go see this in manchester. <http://www.goseethis.com/> (visited on 08-05-2011).
- [30] Hadoop. Libraries for reliable, scalable, distributed computing. <http://hadoop.apache.org/> (visited on 10-08-2011).
- [31] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, CSCW '00, pages 241–250, New York, NY, USA, 2000. ACM.
- [32] Jim Highsmith and Martin Fowler. The agile manifesto. *Software Development Magazine*, 9(8):29–30, 2001.
- [33] IETF. Rfc4287: The atom syndication format. <http://tools.ietf.org/html/rfc4287> (visited on 09-05-2011), December 2005.
- [34] IETF. Rfc5545: Internet calendaring and scheduling core object specification (icalendar). <http://tools.ietf.org/html/rfc5545> (visited on 09-05-2011), September 2009.

- [35] IKnow. Information forensics - smart indexing. <http://www.iknow.be/> (visited on 08-05-2011).
- [36] IPTC International Press Telecommunications Council. Eventsml-g2 standard. http://www.iptc.org/site/News_Exchange_Formats/EventsML-G2/ (visited on 08-05-2011).
- [37] Mehmet Kayaalp, Tansel Ozyer, and Sibel Ozyer. A mash-up application utilizing hybridized filtering techniques for recommending events at a social networking site. *Social Network Analysis and Mining*, pages 1–9, 2010. 10.1007/s13278-010-0010-8.
- [38] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. GroupLens: applying collaborative filtering to usenet news. *Commun. ACM*, 40:77–87, March 1997.
- [39] Yehuda Koren and Robert Bell. *Recommender Systems Handbook*, chapter Chapter 5: Advances in Collaborative Filtering, pages 145–186. Springer, 2011.
- [40] Last.fm. <http://www.last.fm> (visited on 08-05-2011).
- [41] Danielle Hyunsook Lee. Pittcult: trust-based cultural event recommender. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 311–314, New York, NY, USA, 2008. ACM.
- [42] Daniel Lemire and Anna Maclachlan. Slope one predictors for online Rating-Based collaborative filtering. In *Proceedings of SIAM Data Mining (SDM'05)*, 2005.
- [43] James R. Lewis and Jeff Sauro. The factor structure of the system usability scale. In *Proceedings of the 1st International Conference on Human Centered Design: Held as Part of HCI International 2009*, HCD 09, pages 94–103, Berlin, Heidelberg, 2009. Springer-Verlag.
- [44] Li-Hua Li. A personalized local event recommendation system for mobile users. Master's thesis, Chaoyang University of Technology, 2006.
- [45] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7:76–80, January 2003.
- [46] Mahout. Scalable machine learning and data mining libraries. <http://mahout.apache.org/> (visited on 10-08-2011).
- [47] Mashical. Mash ical - calendar aggregator. <http://www.mashical.com/> (visited on 10-08-2011).

- [48] Prem Melville and Vikas Sindhwani. Recommender systems. In *Encyclopedia of Machine learning*. Springer, 2010.
- [49] MicroFormats. hcalendar 1.0 specs. <http://microformats.org/wiki/hcalendar> (visited on 09-05-2011).
- [50] Einat Minkov, Ben Charrow, Jonathan Ledlie, Seth Teller, and Tommi Jaakkola. Collaborative future event recommendation. In *Proceedings of the 19th ACM international conference on Information and knowledge management, CIKM '10*, pages 819–828, New York, NY, USA, 2010. ACM.
- [51] Miquel Montaner, Beatriz Lopez, and Josep Lluís de la Rosa. A taxonomy of recommender agents on the internet. *Artificial Intelligence Review*, 19:285–330, 2003. 10.1023/A:1022850703159.
- [52] Yashar Moshfeghi. Intelligent rss aggregator. Master’s thesis, Department of Computing Science - University of Glasgow, September 2007.
- [53] Teemu Mutanen, Joni Niemi, Sami Nousiainen, Lauri Seitsonen, and Teppo Veijonen. Cultural event recommendations. a case study. 2008.
- [54] Netflix. Netflix prize. <http://www.netflixprize.com/> (visited on 01-09-2011).
- [55] OpenCalais. <http://www.opencalais.com/> (visited on 08-05-2011).
- [56] Georgios Paliouras, Alexandros Mouzakidis, Vassileios Moustakas, and Christos Skourlas. Pns: A personalized news aggregator on the web. In Maria Virvou and Lakhmi Jain, editors, *Intelligent Interactive Systems in Knowledge-Based Environments*, volume 104 of *Studies in Computational Intelligence*, pages 175–197. Springer Berlin / Heidelberg, 2008.
- [57] Marco de Gemmis Pasquale Lops and Giovanni Semeraro. *Recommender Systems Handbook*, chapter Chapter 3: Content-based Recommender Systems: State of the Art and Trends, pages 73–106. Springer, 2011.
- [58] Michael J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artif. Intell. Rev.*, 13:393–408, December 1999.
- [59] P. Perny and J. D. Zucker. Preference-based search and machine learning for collaborative filtering: the "film-conseil" movie recommender system. *Revue I3*, pages 1–40, 2001.
- [60] Plancast. <http://plancast.com/> (visited on 09-05-2011).
- [61] Daniele Quercia, Neal Lathia, Francesco Calabrese, Giusy Di Lorenzo, and Jon Crowcroft. Recommending social events from mobile phone location data. *Data Mining, IEEE International Conference on*, 0:971–976, 2010.

- [62] Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40:56–58, March 1997.
- [63] Elaine Rich. User modeling via stereotypes. *Cognitive Science*, 3(4):329–354, 1979.
- [64] Ruby. A programmer’s best friend. <http://www.ruby-lang.org/> (visited on 01-09-2011).
- [65] RubyOnRails. Web development that doesn’t hurt. <http://rubyonrails.org/> (visited on 01-09-2011).
- [66] G. Salton. *The SMART Retrieval System - Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.
- [67] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18:613–620, November 1975.
- [68] Gerard Salton. *Automatic Text Processing*. Addison-Wesley Publishing Company, 1989.
- [69] Juan J. Samper, Pedro A. Castillo, Lourdes Araujo, J.J. Merelo, Oscar Cordon, and Fernando Tricas. Nectarss, an intelligent rss feed reader. *Journal of Network and Computer Applications*, 31(4):793 – 806, 2008.
- [70] Jeff Sauro. Measuring usability with the system usability scale (sus). <http://www.measuringusability.com/sus.php> (visited on 31-08-2011), February 2011.
- [71] J. Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, chapter 9, pages 291–324. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2007.
- [72] Barry Schwartz. *The paradox of choice : why more is less*. ECCO, New York :, 1st ed. edition, 2004.
- [73] Upendra Shardanand and Pattie Maes. Social information filtering: algorithms for automating "word of mouth". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '95, pages 210–217, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.
- [74] Alvin. Toffler. *Future shock*. Pan London, 1971.
- [75] Upcoming. Local event guide. <http://upcoming.yahoo.com/> (visited on 09-05-2011).

- [76] Upcoming. Restful developer api. <http://upcoming.yahoo.com/services/api/> (visited on 01-09-2011).
- [77] W3C. Geolocation api specification. <http://www.w3.org/TR/geolocation-API/> (visited on 07-05-2011), September 2010.
- [78] Xampp. Apache friends: xampp solution. <http://www.apachefriends.org/en/xampp.html> (visited on 15-08-2011).
- [79] Nuria Oliver Xavier Amatriain, Alejandro Jaimes and Josep M. Pujol. *Recommender Systems Handbook*, chapter Chapter 2: Data Mining Methods for Recommender Systems, pages 39–72. Springer, 2011.

Appendix A

Evaluation questionnaires

A.1 Initial form

Picalag: evaluation of an intelligent Manchester events aggregator. Initial form

This study is evaluating a web-application developed as part of a MSc project. The proposed system is a cultural events calendar aggregator with a recommendation module. With this study, we want to test if such a system is useful to assist people looking for events to attend for a day or a night out in Manchester or willing to broaden their interest with new experiences recommended by the system.

The participants will be invited to fill a first questionnaire to collect information about similar systems they are used to use in a daily basis. They will then be able to create an account on the website and test it as if they were organizing a day or night out for around 5 to 10 minutes each day during 6 days. Each session will end with a short questionnaire to evaluate the relevance of the information the system provided to the user. At the end of the study (6th day), a final questionnaire will be presented to the user to evaluate the usability of the system and collect feedbacks to enable future work to be suggested or changes to be made on the system.

All in all, participants are not expected to spend more than 1 hour of their time for this study. It is not anticipated that there will be any physical discomfort associated with the study, as it is normal web browsing. Participants will be free to take a break or withdraw at any point.

If you decide to participate in this research, you will not be paid and no out-of-pocket expenses will be required from you. The whole study takes place remotely so you can participate from any computer connected to the Internet at any time suitable for you.

Data will be stored in a secure place and be made anonymous so that no one will be able to recognize who the data belongs to.

To participate to this study you must:

- Be over 18
- Have a basic level of English (to answer questionnaires and use the system)
- Be currently living in Manchester (or you should have spent some time there in the past)

Before you start, please read the Information sheet provided as a PDF file <http://eleves.mines.inpl-nancy.fr/~grill6013/infosheet.pdf>.

If you want to receive further information or ask any question before or during the experiment please contact Sébastien GRILLOT (sebastien.grillot@cs.man.ac.uk).

Ethics approval number CS7.

Consent form

If you are happy to participate to this study please complete the consent form below.

– Information *

☐ I confirm that I have read the attached information sheet on the above project and have had the opportunity to consider the information and ask questions and had these answered satisfactorily.

– Participation *

☐ I understand that my participation in the study is voluntary and that I am free to withdraw at any time without giving a reason.

– Use of data *

☐ I agree to the use of anonymous quotes.

– Requirements *

☐ I am over 18 I am living or I stayed in Manchester

– I agree to take part in the above project. Name of participant: *

Demographic

These questions are used to know a bit more about you.

– Age *

– Gender *

☐ Male

☐ Female

– Occupation

Are you a student? A professional? ...

Web habits

These questions are designed to let us know a bit more about your habits in using computers and web technologies.

– How would you self describe your knowledge about computing ?

If it is the first time you see a computer in your whole life and you do not even know how to answer this question, click (with your mouse) the grey circle under the 1 below this text. If you are a regular computer user, you are probably in the middle of this scale. If you study or work in computer science, you are probably a 5 (or a 4 if you are modest or do not want us to think you are a geek).

	1	2	3	4	5	
Computer illiterate	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Computer literate

– Web usage

How often do you use the web ?

☐ Never (if you are here you are probably a liar)

☐ Once a week

☐ Once a day

☐ Less than one hour each day

☐ Between one and three ours each day

☐ More than three hour per day

– What do you know about aggregators ?

Wikipedia: "Aggregator refers to a web site or computer software that aggregates a specific type of information from multiple online sources"

☐ Aggrega-what ? I did not know what an aggregator is

☐ I know what it is but I have never used it

☐ I know what it is and I use some of them

☐ I know what it is and I use them all the time

– If you use aggregators. What kind of content do you usually aggregates?

☐ News (e.g. Google Reader or any advanced RSS reader)

☐ Events

☐ Videos

☐ Other:

– Do you manage you calendar thanks to a web application or a normal application ?

There are plenty of calendar application out there (e.g. Google Calendar, your favorite smartphone calendar app). Some people use sometimes social networks such as events on Facebook to manage their time: in this case you could click "other". Some people use paper calendars or no calendar at all.

☐ yes always

☐ no never

☐ Other:

– Do you use the web as a source of inspiration or information when you are looking for an event to attend ?

☐ I use it as a source of inspiration to discover new events or venues

☐ I use it as a source of information to know what happens in my favorite venues

☐ Both

☐ No

– In the case you use the web, which service(s) do you usually use ?

This is a list of online events websites. If you favorite services are not in the list, feel free to add yours in the "other" field

☐ Upcoming

☐ City Life

- ☐ Facebook
- ☐ Eventbrite
- ☐ Go see this
- ☐ I go directly on my favorite venues websites
- ☐ Other:

– How many different sources do you usually visit when you are preparing a day or night out ?

- ☐ only 1
- ☐ 2-4
- ☐ 4-10
- ☐ more than 10

– Do you usually use recommender systems ?

There are recommender systems everywhere on the web (e.g. Amazon, Youtube). Do you use them ? what do you think of them ?

- ☐ No, because I do not trust the recommendations
- ☐ No, because the recommendations are usually not useful
- ☐ No, because they always recommend the same things
- ☐ Yes, I use them
- ☐ Yes, I am a fan of these systems

– What do you expect from recommender systems ?

- ☐ To make me save time
- ☐ To stick to my interest and recommend only things that should interest me
- ☐ To broaden my interest and surprise me sometimes
- ☐ To be 100% accurate
- ☐ Nothing in particular
- ☐ Other:

– What kind of recommendations do you trust more ?

- ☐ These items have similar content as the one you are browsing
- ☐ These items match your profile
- ☐ Similar users also visited/bought...

- ☐ Most popular items
- ☐ Random items
- ☐ Those who visited/purchased this item also visited/purchased...
- ☐ Other:

A.2 Picalag evaluation - End of session Form

After you spent 5 to 10 minutes on Picalag looking at the events and recommendations, please complete this short form to give feedbacks about your experience.

If you do not know what to answer or if you need more time with the system to be able to give an answer, leave the question blank.

- How long (in days, including today) have you been using the system ? *

1	2	3	4	5	6	7	8	9	10
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

- What are your favorite recommendation modules for this session ?

- ☐ On My Picalag page, most popular events
- ☐ On My Picalag page, random events
- ☐ On My Picalag page, events matching my profile
- ☐ On My Picalag page, recommendation based on other users' ratings
- ☐ On Favorite venue page, most popular venues
- ☐ On Favorite venue page, recommended venues based on others
- ☐ On an event page, similar events based on content
- ☐ On an event page, similar events based on other users' ratings

- The recommended items I received through the system were relevant

tell us if you agree this statement. 1 = Strongly disagree ; 2 = Disagree ; 3 = Neither agree nor disagree ; 4 = Agree ; 5 = Strongly agree

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

- The recommended items I received through the system matched my interest

tell us if you agree this statement. 1 = Strongly disagree ; 2 = Disagree ; 3 = Neither agree nor disagree ; 4 = Agree ; 5 = Strongly agree

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

- I understand why the items were recommended by the system Given your interest and the items you browsed/like/disliked, do you think the system was able to recommend events that were similar ?

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

- While using the system I felt in control

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

- While using the system I felt comfortable

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

- While using the system I felt confident

Do you trust the system?

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

- Was the interface effective for solving the task ?

You can check multiple answers if you agree with them

- ☐ The interface was effective for solving the task
- ☐ The interface was NOT effective for solving the task
- ☐ It helped me to give feedback easily
- ☐ It helped me to see new recommended events or venues
- ☐ It helped me to find relevant events or venues
- ☐ Other:

- How helpful was the system during this session ?

	1	2	3	4	5	
Not useful at all	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very useful

- Can you give a grade from 1 to 10 for the system during this session ?

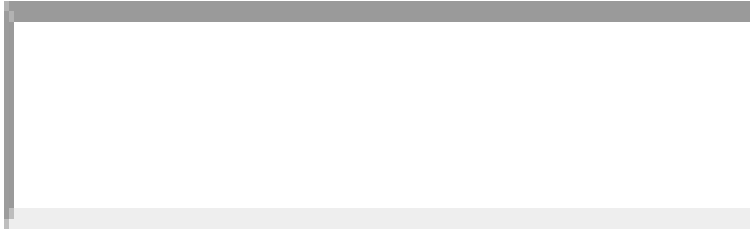
1	2	3	4	5	6	7	8	9	10
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

- What do you think ?

- ☐ I would like to see Picalag as my default website source for events in Manchester
- ☐ The system made me save time

- Any comment ?

leave your feedback if you want



A.3 Picalag Evaluation: Final form

This final form is designed to collect your feedback after using the system several days.

System usability

What do you think about the system and its interface ?

– I think that I would like to use this system frequently

tell us if you agree this statement. 1 = Strongly disagree ; 2 = Disagree ; 3 = Neither agree nor disagree ; 4 = Agree ; 5 = Strongly agree

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

– I found the system unnecessarily complex

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

– I thought the system was easy to use

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

– I think that I would need the support of a technical person to be able to use this system

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

– I found the various functions in this system were well integrated

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

– I thought there was too much inconsistency in this system

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

– I would imagine that most people would learn to use this system very quickly

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree
– I found the system very cumbersome to use						
	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree
– I felt very confident using the system						
	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree
– I needed to learn a lot of things before I could get going with this system						
	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

About your experience

- Can you give a grade to the system for the quality of recommendations you received ?

1	2	3	4	5	6	7	8	9	10
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

- Did you feel the quality of recommendations was improving during the week as the system learnt your interests ?

	1	2	3	4	5	
Not at all	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	For sure

- If this system comes out as a real service would you use it ?

☐ Yes

☐ No

- What do you think ?

You can check multiple answers if you agree with them

☐ The service is useful

☐ The service makes me save time

☐ I am satisfied with the captured interest

☐ I think the system could replace all my other sources of information for public events (at least many of them)

☐ I am not sure such a service can be useful

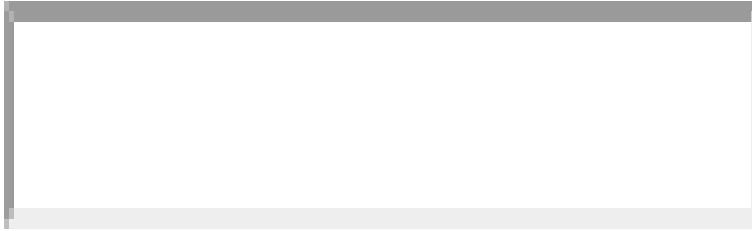
☐ It really has to be improved before it can be a real service

☐ The captured interest visibly did not match my tastes, I am disappointed

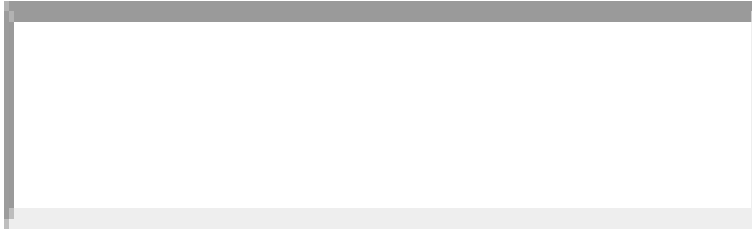
– Which features were the most useful ?

- ☐ Mark venues as favorite
- ☐ List all events taking place on my favorite venues on a single page
- ☐ Recommend new venues I could enjoy
- ☐ On my picalag, Recommend events based on my profile
- ☐ On my picalag, Recommend events based on what others rated
- ☐ On my picalag, list the most popular events
- ☐ On my picalag, give a random list of events to make me discover new things
- ☐ On event page, recommend similar events
- ☐ Rating the events
- ☐ Export to my calendar programs
- ☐ Export to my social networks
- ☐ Search box
- ☐ Tags and tag cloud

– What did you like about the system ?



– What did you dislike about the system ?

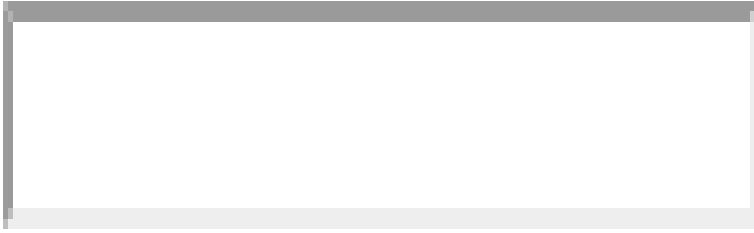


– How responsive was the system ?

Did you have the feeling you had to wait ages to get your recommendations or any pages ?

	1	2	3	4	5	
Very slow	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very fast

– Any other comment ?



Appendix B

Documentation

B.1 Picalag Front-end application

B.1.1 Installation instructions

Setup from scratch

You will need to:

- Install git, a distributed version control system. Read the Github Git Guides to learn how to use git.
- Install Ruby, a programming language. You can use MRI Ruby 1.8.7, or Phusion REE (Ruby Enterprise Edition). Your operating system may already have it installed or offer it as a pre-built package.
- Install RubyGems 1.3.x or newer, a tool for managing software packages for Ruby.
- Install SQLite3, a database engine. Your operating system may already have it installed or offer it as a pre-built package.
- Install Bundler, a Ruby dependency management tool. You should run `gem install bundler` as root or an administrator after installing Ruby and RubyGems.
- Checkout the source code. Run:

```
git clone git://github.com/picalag/calagator.git
```

which will create a calagator directory with the source code. Go into this directory and run the remaining commands from there.
- Install Bundler-managed gems, the actual libraries that this application uses, like Ruby on Rails. You should run `bundle`, which may take a long time to complete.

- Modify themes/picalag/settings.yml file (GPS origin, timezone, PServer URL, title...)
- Optionally setup API keys for external services so that maps will be displayed, see the API Keys section for details.
- Start the search service if needed, see the Search engine section for details.

Development

To run Calagator in development mode, which automatically reloads code as you change it:

- Follow the Setup instructions above.
- Initialize your database, run
`bundle exec rake db:migrate db:test:prepare`
- Start the Ruby on Rails web application by running `./script/server` (UNIX) or `ruby script/server` (Windows).
- Open a web browser to `http://localhost:3000/` to use the development server
- Read the Rails Guides to learn how to develop a Ruby on Rails application.
- When done, stop the Ruby on Rails server `script/server` by pressing CTRL-C.

Production

To run Calagator in production mode, which runs more quickly (cache), but doesn't reload code:

- Follow the Setup instructions above.
- Don't forget to do things like modify the theme settings and secrets files.
- Setup a firewall to protect ports used by your search engine, see the Search engine section for details.
- Initialize your database, run
`bundle exec rake RAILS_ENV=production db:migrate`
- Run `bundle exec rake tmp:cache:clear` to clear your cache after updating your application's code.
- Setup a production web server using Phusion Passenger, Thin, Rainbows, etc. These will be able to serve more users more quickly than `script/server`.

Security and secrets.yml, API Keys, Search engine

See `INSTALL.md` file for details

Setup using TurnKey Linux Ruby on Rails distribution

To make the installation even simpler, one can use TurnKey Linux Ruby On Rails distribution available at <http://www.turnkeylinux.org/rails>. For quick start guide, see <http://www.sharingatwork.com/2009/10/get-started-building-web-apps-with-your-own-ruby-on-rails-virtual-development-server/>.

It comes with MySQL by default. Picalag is compatible with this RDBMS but uses SQLite3 by default.

To use MySQL instead, modify `config/database.yml` file.

To use SQLite3, simply install it with apt-get and gem:

```
apt-get install libsqlite3-dev sqlite3
gem install sqlite3-ruby
```

To deploy Picalag in a production environment, follow instructions on <http://www.turnkeylinux.org/docs/rails/deployment>. If you are using SQLite, you may have some access mode issue (use `chmod` command).

B.1.2 Import RESTful API

Only one method is opened for event import. Each request posts one event and the corresponding venue.

POST ... /sources/API_import_event

Parameters:

Title (string) Event title

Link (string) URL to Event page (Source URL)

This URL is a unique identifier for the event in Picalag. If an event already exists with the same URL, it will be overwritten.

Description (string) Event description text

Start (timestamp) Event start (format: YYYY-MM-dd HH:mm)

End (timestamp) Event end (format: YYYY-MM-dd HH:mm)

Category (string) Event category

Venue (string) Venue name

VenueInfo (string) Venue info description

Tel (string) Venue Tel number

Address (string) Venue address

Longitude (double) Venue longitude

Latitude (double) Venue latitude

LinkVenue (string) URL to Venue page

This URL is a unique identifier for the venue in Picalag. If two events take place in a venue with this URL they will be added to one single venue in Picalag.

B.2 PServer Back-end application

B.2.1 Installation instructions

Install and configure Java SDK, Glassfish and MySQL.

Get sources from <http://github.com/picalag>.

Database

Create a database “picalag_pserver” and load the structure `picalag_pserver.sql` file.

If it is not done already, configure and start a GlassFish domain.

Setup connection pool following tutorial on <http://www.albeesonline.com/blog/2008/08/06/creating-and-configuring-a-mysql-datasource-in-glassfish-application-server/>. The name of the resource is “jdbc/picalag_pserver” (this can be changed on `web.xml` config file in sources).

Source compilation

ActiveJDBC requires class Instrumentation to work (it adds some magical byte code to the models after compilation). Details about this step can be found in this project’s wiki: <http://code.google.com/p/activejdbc/wiki/Instrumentation>.

If NetBeans is used as IDE, a simple integration method is given on this page: <http://code.google.com/p/activejdbc/wiki/NetbeansIntegration>.

Once compiled as a Java web application, an archive *.war should be generated, containing the deployable program including all dependencies (size around 19Mb).

Deployment

To deploy the application in Glassfish, open the admin panel in a web browser. On the menu choose “application” and select the “*.war” archive to deploy it. That is all, the application is running. You can try to call the /API/ping method to check it is working (see API documentation below).

B.2.2 RESTful API

All these methods can be accessed either with a GET or a POST request.

/API/post_event

Adds a new event to PServer DB

Parameters:

id_event (int) event ID in calagator

title (string) event title in calagator

description (string) event description in calagator

venue (string) venue name in calagator

id_venue (int) venue ID in calagator

tags (string) event tags (delimited with “,”)

date (string) event start date: format "yyyy-mm-dd" or "yyyy-mm-dd hh:mm"

/API/post_venue

Adds a new venue to PServer DB

Parameter:

id_venue (int) venue ID in calagator

/API/add_user

Creates a new user or updates an existing user in PServer DB

Parameters:

id_user (int) user ID in calagator

male (boolean) "true" if male, "false" if female

dob (string) date of birth: format "yyyy-mm-dd" or "yyyy-mm-dd hh:mm"

/API/rate_event

Called when a user rates an event

Parameters:

id_user (int) user ID in calagator

id_event (int) event ID in calagator

rating (mixed) rating value

Possible types for *rating*:

(int) an explicit rating

(string) based on an implicit rating event ("disliked", "neutral", "viewed", "added", "liked", "shared")

/API/view_event

A user views an event: both event feature vector and user profile should be updated

Parameters:

id_user (int) user ID in calagator

id_event (int) event ID in calagator

/API/never_again

Removes the event features from user's profile and rate it as disliked event

Parameters:

id_user (int) user ID in calagator

id_event (int) event ID in calagator

/API/add_venue_to_favorite

Adds a venue to user's favorite venues list

Parameters:

id_user (int) user ID in calagator

id_venue (int) venue ID in calagator

/API/del_venue_from_favorite

Removes a venue from user's favorite venues list

Parameters:

id_user (int) user ID in calagator

id_venue (int) venue ID in calagator

/API/get_favorite_venues

Returns an XML with user's favorite venues list of calagator IDs

Parameter:

id_user (int) user ID in calagator

Ouput:

```
<picalag>
  <favorite_venues>
    <venue>130</venue>
    ...
  </favorite_venues>
</picalag>
```

/API/get_recommendations_CB_event

Returns an XML with recommended events by a content-based recommender system.

Recommendations are events similar to *id_event* event

Parameters:

id_event (int) event ID in calagator

nb_recs (int) *optional* maximum number of recommendations (default = 5)

date (string) *optional* date of recommended events: format "yyyy-mm-dd" or "yyyy-mm-dd hh:mm" (default = same date as the *id_event* event)

Output:

```
<picalag>
  <recommendations>
    <event>
      <id>38</id>
      <distance>0.7999999999999998</distance>
    </event>
    ...
  </recommendations>
</picalag>
```

/API/get_recommendations_CB_user

returns an XML with recommended events by a content-based recommender system. Recommendations are events similar to *id_user* user's profile

Parameters:

id_user (int) user ID in calagator

date (string) date of recommended events: format "yyyy-mm-dd" or "yyyy-mm-dd hh:mm"

nb_recs (int) *optional* maximum number of recommendations (default = 5)

Output:

```
<picalag>
  <recommendations>
    <event>
      <id>38</id>
      <distance>0.7999999999999998</distance>
```

```

        </event>
        ...
    </recommendations>
</picalag>

```

/API/get_recommendations_CF_event

Returns an XML with recommended events by a collaborative filtering recommender system (ItemBased). Recommendations are events similar to **id_event** event

Parameters:

id_event (int) event ID in calagator

nb_recs (int) **optional** maximum number of recommendations (default = 5)

date (string) **optional** date of recommended events: format "yyyy-mm-dd" or "yyyy-mm-dd hh:mm" (default = same date as the **id_event** event)

Output:

```

<picalag>
  <recommendations>
    <event>
      <id>38</id>
      <distance>0.7999999999999998</distance>
    </event>
    ...
  </recommendations>
</picalag>

```

/API/get_recommendations_CF_user

Returns an XML with recommended events by a collaborative filtering recommender system (UserBased). Recommendations are events similar to **id_user** user's neighbourhood interest

Parameters:

id_user (int) user ID in calagator

date (string) date of recommended events: format "yyyy-mm-dd" or "yyyy-mm-dd hh:mm"

nb_recs (int) *optional* maximum number of recommendations (default = 5)

Output:

```
<picalag>

  <recommendations>

    <event>

      <id>38</id>
      <neighbour_rating>1</neighbour_rating>
      <neighbour_rating>5</neighbour_rating>
      ...
    </event>
    ...

  </recommendations>

</picalag>
```

/API/get_recommendations_venues

Returns an XML with recommended venues by a collaborative filtering recommender system (SlopeOne). Recommendations are venues similar to *id_user* user's favorite venues

Parameters:

id_user (int) user ID in calagator

nb_recs (int) *optional* maximum number of recommendations (default = 5)

Output:

```
<picalag>

  <recommendations>

    <venue>
```



```

        <id>320</id>
        <similar>405</similar>
        <similar>245</similar>
        ...
    </venue>
    ...
</recommendations>
</picalag>

```

`/API/get_recommendations_most_popular_events`

Returns an XML with the most popular events (according to the sum of ratings)

Parameters:

date (string) date of recommended events: format "yyyy-mm-dd" or "yyyy-mm-dd hh:mm"

nb_recs (int) *optional* maximum number of recommendations (default = 5)

id_user (int) *optional* user ID in calagator, if given, events known by this user won't be recommended

Output:

```

<picalag>
  <recommendations>
    <event>
      <id>1</id>
      <grade>5</grade>
    </event>
    ...
  </recommendations>
</picalag>

```

`/API/get_recommendations_most_popular_venues`

Returns an XML with the most popular venues (according to the number of user having them on their favorite list)

Parameters:

nb_recs (int) *optional* maximum number of recommendations (default = 5)

id_user (int) *optional* user ID in calagator, if given, venues already in this user's favorite list won't be recommended

Output:

```
<picalag>
  <recommendations>
    <venue>
      <id>1</id>
      <fans>10</fans>
    </venue>
    ...
  </recommendations>
</picalag>
```

`/API/get_recommendations_random_events`

Returns an XML with random events (according to the sum of ratings)

Parameters:

date (string) date of recommended events: format "yyyy-mm-dd" or "yyyy-mm-dd hh:mm"

nb_recs (int) *optional* maximum number of recommendations (default = 5)

id_user (int) *optional* user ID in calagator, if given, events known by this user won't be recommended

Output:

```
<picalag>
  <recommendations>
```

```

        <event>
            <id>38</id>
        </event>
        ...
    </recommendations>
</picalag>

```

/API/get_recommendations_random_venues

Returns an XML with random venues (according to the number of user having them on their favorite list)

Parameters:

nb_recs (int) *optional* maximum number of recommendations (default = 5)

id_user (int) *optional* user ID in calagator, if given, venues already in this user's favorite list won't be recommended

Output:

```

<picalag>
    <recommendations>
        <venue>
            <id>1</id>
        </venue>
        ...
    </recommendations>
</picalag>

```

/API/is_favorite_venue

Returns "true" if the venue *id_venue* is one of *id_user*'s favorite

Parameters:

id_user (int) user ID in calagator

id_venue (int) venue ID in calagator

/API/get_rating

Returns user's rating for an event Parameter: id_user (int) user ID in calagator

Parameter:

id_event (int) event ID in calagator

/API/ping

Sends OK status if server is running