Conception and Development of a Social Web Product — Spateo

Mário Lopes

Report of Project/Dissertation
Masters in Informatics and Computational Engineering
Advisor and Supervisor: Prof. Ademar Aguiar

7th July 2008
© Mário Lopes, 2008
Conception and Development of a Social Web Product — Spateo

Mário Lopes

Report of Project/Dissertation
Masters in Informatics and Computational Engineering

Approved in oral examination by the committee:

Chair: Ana Paula Cunha da Rocha (Prof./FEUP)

External Examiner: António Rito Silva (Prof./IST)

16th July, 2008
CONFIDENTIALITY NOTICE

This document contains information confidential and proprietary to its author, MÁRIO LOPES. The information may not be used, disclosed or reproduced without the prior written authorization of MÁRIO LOPES and those so authorized may only use the information for the purpose of evaluation consistent with authorization. Reproduction of any section of this document must include this legend. Unauthorized use of any information enclosed in this document is strictly prohibited and action will be taken to the full extent of law.
To all the entrepreneurs that continuously pursue the dream of changing the world, one company at a time.
Abstract

The Internet has become the revolution of the twenty first century. From connecting people physically apart to creating new business opportunities, the Internet is now the platform where several billions of dollars flow every day and night. What started as a military research project has become a mainstream commodity throughout all developed countries, imminently reaching underdeveloped countries to help them strive.

Within the world wide web, other minor yet important revolutions also took place. Social networks changed the way people virtually connect to each other and share their information with family, friends or even strangers. Paired with the commoditization of the Internet, social networks and applications alike started playing a major role offset the virtual realm and into people’s lives. Assisting on real life tasks has been one of the immense advantages of the Internet.

One particular real life activity that can take advantage of the Internet is that of event planning. Events are a very humane consequence of human and social beings living together in society. Individuals tend to gather together with friends or family to watch sports, attend a rock concert or go to the cinema. The Internet is capable of leveraging the effort required to individuals to organize each other and plan their events more efficiently.

In this thesis we propose a web application that mixes a social platform and an events’ application in order to allow people to plan, organize and advertise their events and, at the same time, connect to people that are interested in the same kind of events. Furthermore, we make available a platform that clearly distinguishes the type of events and provides tools accordingly.

From a technical perspective, we set as a goal the conception and development of such system in terms of functional and non-functional requirements, as well as verifying and validating that the application is working as expected. From a business point of view, and considering that the tools created by the mastery of engineering are of no use if not sold and put to use, we establish a very clear goal that the plain success of this project depends on the market acceptability and commercial success of the project, notwithstanding its technical merit.
Acknowledgments

Full acknowledge goes to all entrepreneurs and businessman that put their mental and physical ability to use and produce value. Recognition is given to all the working class that tries to exceed itself and trade for money only the result of what their best ability can achieve. Cognizance to all the individuals and brilliant minds that thrive by being the best.

Also, reference has to be made to all the individuals like Prof. Ademar Aguiar that have decided to use their mental brilliance as the tool to spread only but the greatest thoughts. He has been a mentor like very few are.

A word of appreciation to my business partner Nuno Baldaia, a seldom case of excellency allied with perseverance and will to produce only the best his work can accomplish.

Finally, no acknowledgment would ever be truthful if no reference was to be made to all the people that share with us the joy of living, teach us to excel ourselves and make us better human beings. To my family, friends and more than friends, my deepest recognition to what you represent to me. Thank You.
# Contents

1 Introduction 1
   1.1 Objectives ............................................ 4
   1.2 Motivation ............................................ 5
   1.3 Social Networks ....................................... 6
   1.4 Event Planning ....................................... 7

2 State of Art 8
   2.1 The Social Web ........................................ 9
      2.1.1 Social Networks .................................... 9
      2.1.2 Events Planning and Management ................ 10
   2.2 Comparison ........................................... 12
      2.2.1 A. Social Features ................................. 12
      2.2.2 B. Events ......................................... 14
      2.2.3 C. Venues and Service Providers ................ 15
      2.2.4 D. Event Browsing ................................ 16
      2.2.5 E. Localization .................................... 16

3 Spateo - Conception and Development 17
   3.1 Functional Requirements .............................. 18
      3.1.1 Events Management ................................. 19
      3.1.2 Events Browsing .................................... 21
      3.1.3 Social Network ..................................... 22
      3.1.4 User and Profile Management ..................... 24
   3.2 Non-Functional Requirements ........................ 25
   3.3 Timeline and Expected Results ........................ 25
   3.4 Development .......................................... 27
      3.4.1 Planning ........................................... 27
      3.4.2 Development Methodology .......................... 28
      3.4.3 Architecture ....................................... 29
      3.4.4 Design Patterns .................................... 32
      3.4.5 Technologies ....................................... 34
         3.4.5.1 Database ...................................... 35
         3.4.5.2 Full Text Search ............................... 36
         3.4.5.3 Storage ........................................ 36
         3.4.5.4 Web Development Framework ................... 37
         3.4.5.5 Web Server ..................................... 38
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.5.6</td>
<td>Application Server</td>
<td>38</td>
</tr>
<tr>
<td>3.4.5.7</td>
<td>Operating System</td>
<td>38</td>
</tr>
<tr>
<td>3.4.6</td>
<td>Data Storage</td>
<td>39</td>
</tr>
<tr>
<td>3.4.7</td>
<td>Testing</td>
<td>40</td>
</tr>
<tr>
<td>3.4.8</td>
<td>Usability and User Experience</td>
<td>41</td>
</tr>
<tr>
<td>3.4.9</td>
<td>Localization (i13n) and Internationalization (i18n)</td>
<td>42</td>
</tr>
<tr>
<td>3.5</td>
<td>Generalization</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>Prototype</td>
<td>54</td>
</tr>
<tr>
<td>4.1</td>
<td>Frontpage</td>
<td>55</td>
</tr>
<tr>
<td>4.2</td>
<td>Session management</td>
<td>57</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Signup</td>
<td>57</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Login</td>
<td>58</td>
</tr>
<tr>
<td>4.3</td>
<td>Social features</td>
<td>59</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Profile</td>
<td>59</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Albums/Photos</td>
<td>60</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Friends</td>
<td>61</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Inbox</td>
<td>62</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Compose a message</td>
<td>63</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Groups</td>
<td>64</td>
</tr>
<tr>
<td>4.4</td>
<td>Events Platform</td>
<td>65</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Create an event</td>
<td>65</td>
</tr>
<tr>
<td>4.4.1.1</td>
<td>Manage the event</td>
<td>70</td>
</tr>
<tr>
<td>4.4.1.2</td>
<td>Event page</td>
<td>71</td>
</tr>
<tr>
<td>4.4.1.3</td>
<td>Event RSVP</td>
<td>72</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Search</td>
<td>73</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Themes</td>
<td>74</td>
</tr>
<tr>
<td>4.4.3.1</td>
<td>Theme creation</td>
<td>75</td>
</tr>
<tr>
<td>5</td>
<td>Conclusions and Further Work</td>
<td>76</td>
</tr>
</tbody>
</table>

## Bibliography | 78

### A Analysis and Specification of Functional and Non-Functional Requirements, and Test Policies of the “Spateo” Application | 79
List of Figures

1.1 The Long Tail theory. .................................................. 2
1.2 An example of a social network diagram. ................................ 6

3.1 Block diagram of the Spateo web application. .......................... 20
3.2 Events management use case diagram. ..................................... 21
3.3 Events browsing use case diagram. ....................................... 22
3.4 Social Network use case diagram. ......................................... 23
3.5 User and Profile Management use case diagram. .......................... 24
3.6 Development and product launch roadmap. ................................ 26
3.7 Typical three-tier architecture for a web application [wik(2008)]. ....... 30
3.8 Physical architecture of Spateo. ........................................... 31
3.9 Ruby on Rails internal logic architecture [rub(2008)]. .................... 32
3.10 The Model View Controller object diagram. .............................. 33
3.11 A concrete example of the Model View Controller pattern. .............. 34
3.12 A database example schema for storing information related to the events. 44
3.13 Class inheritance to support event types and properties. ................. 45
3.14 The Dynamic Object Model class diagram. .............................. 46
3.15 The Dynamic Object Model class diagram applied to the events. ......... 47
3.16 The database schema to accommodate the Dynamic Object Model. ...... 51

4.1 Entrance page to the Spateo web application. ............................ 55
4.2 Create an user account. .................................................. 57
4.3 Log in on Spateo. ....................................................... 58
4.4 Profile page of a registered user. ........................................ 59
4.5 A photo album. ................................................................ 60
4.6 Friends listing. .................................................................. 61
4.7 The message center where users can read incoming messages, compose new messages and check for invitations. ......................... 62
4.8 “Compose a new message” form. .......................................... 63
4.9 A page of a group. .......................................................... 64
4.10 Create an event - step 1 of 5. ............................................. 65
4.11 Create an event - step 2 of 5. ............................................. 66
4.12 Create an event - step 3 of 5. ............................................. 67
4.13 Create an event - step 4 of 5. ............................................. 68
4.14 Create an event - step 5 of 5. ............................................. 69
4.15 Manage an event. .......................................................... 70
List of Figures

4.16 Event page. ................................................................. 71
4.17 RSVP to an event. .................................................... 72
4.18 Search for other people. .............................................. 73
4.19 Pick a theme for the profile page or for an event. .......... 74
4.20 Create a new theme. ................................................... 75
2.2  Table comparing the most important social websites currently operational. The cross (x) denotes a feature that’s available and a dot (.) not available up to the time of writing. ................................. 13
1 Introduction


While consulting for the European Center of Nuclear Research, Tim Berners-Lee started working on a draft proposing a new way of writing, using and sharing documents amongst CERN researchers. In 1993 Berners-Lee and Dan Connolly finally published a document entitled “Hypertext Markup Language (HTML)” [Berners-Lee and Connolly(1993)] that described how documents could be shared in a standard, open and human-readable way without depending on the machine to interpret binary code. On top of that, the definition of HTML also included the capability of visually formatting a document by including special tags that would be interpreted by a computer.

At the same time the Internet was paving its way, moving from a small in-house research project at the Defense Advanced Research Projects Agency (DARPA) into a global network of computers. By not being tied to a particular and proprietary technology but rather a mesh of new-coming open technologies and standards, the Internet was steadily adopted by several companies and worked by hundreds of researchers around the world. Hypertext, according to the definition of [Berners-Lee and Connolly(1993)], seemed like an efficient way of sharing documents over the soon to be called 'World Wide Web'. Based on the assumption that HTML could become the standard of the Web, the Hypertext Transfer Protocol [Society(1999)] (commonly known as HTTP) emerged, a protocol capable of transferring HTML documents. The HTTP protocol is the underpinnings of the web as we currently know it.

But Tim Berners-Lee could have never predicted that his work and the Internet would become such a milestone in the IT history or have given origin to a market resulting in millions of transactions worth billions of euros per day. What was once a technology that allowed people to remotely connect to each other through their personal computers became a massive business center where people buy and sell goods and services.

The World Wide Web soon spanned over multiple key areas of our contemporary society, bringing to the virtual world what previously required physical presence. Newspapers started maintaining an online version of their publications; banks and other financial institutions allowed people to access and control their accounts without having to stand in front of endless lines of waiting; businesses concocted virtual stores (e-commerce websites) that could sell virtually anything that was available. But besides touching on key-points of our society, the Internet opened up a whole new spectrum of possibilities.
1 Introduction

It didn’t take too long for entrepreneurs to realize that these opportunities could generate money. And even for those not planning on making money on the web, the Internet set the stage for tremendous amateur businesses that had an explosive growth and consequential success. A particular niche that has been experiencing prodigious growth is the one related to social networks that allow people to connect to each other. Social networks and how they allow people to network is briefly touched upon on section 1.3. Companies like News Corp. (through the infamous MySpace), Facebook, Hi5 and many others have been very successful in generating profit out of social websites.

Social Applications meet the Long Tail Theory

The Internet and social networks have also shown that the Internet is not a big, homogeneous community but rather the junction of several small groupings that interact together and usually share similar tastes. Although there are no restrictions to what kind of people you can contact on the Internet, anecdotal observation has shown that people tend to unite with those that have at least some interests in common, acting similarly to what happens in real life. Furthermore, the Internet has shown that some niches that didn’t exist in real life started to gain form in the virtual confinements of the World Wide Web. Actually, the Internet triggered off something that was set to be impossible in real businesses, namely to realize significant profit out of selling small volumes of hard-to-find items to many customers, instead of only selling large volumes of a reduced number of popular items. The group of persons that buy the hard-to-find or "non-hit" items is the customer demographic called the Long Tail [Anderson(2006)]. The Internet sprung this new market. Figure 1.1 depicts the Long Tail theory by drawing a graph of popularity vs selling rate.

![Figure 1.1: The Long Tail theory.](image)

As can be seen from Figure 1.1, the yellow area represents the amount of non-popular items that are sold. Despite its low popularity rate, if we make available a comprehensive offer of non-popular products we might generate the same revenue as selling more
of less products. This observation opened up new opportunities that used to be ignored by entrepreneurs and businessmen as they thought there wouldn’t be a market for their products. In economical terms, profitability can be reduced or even hard to achieve if there aren’t enough consumers. This still holds true. What has changed is the way that producers reach consumers (and vice-versa). The Internet has leveraged this connection and brought closer the goods of the producers and the needs of the consumers. Furthermore, the Internet phenomenon has also shown that there are always consumers to ones’ goods, regardless of how spendthrift or outré a product might be.

Transposing this observation to the web products domain, during the last past years we’ve been witnessing to a lot of websites that started targeting specific and well-defined demographics. Instead of the typical “swiss-army” approach where one would try to conceive a product feature-rich but technically-poor or confusing, web entrepreneurs found it more lucrative to target a well-determined group of people and offer them precisely what they needed and nothing more. This behavior became very apparent in the realm of social applications where one can now find social networks for coffee lovers, wine tasters, dog owners or for pregnant mothers.

The Web as an Events Platform

One particular niche of social websites that has been gaining some traction is one related to events. Despite playing a major role in our life, the Internet hasn’t yet fully replaced some activities that do require live meetings. People have been gathering together for centuries for multiple purposes and they still will do, no matter the motto that brings them together or apart. Social networks can be established by connecting those attending the same events and potentially sharing similar interests. A more detailed view of events networking and how these can mash up with social networks will be described in section 1.4.

In this thesis we draw a proposal for leveraging how people can gather together in real life through the means of an Internet product. The following proposition won’t replace real life meetings (although other distinct products exist for this purpose) but rather make it easier for people to plan and attend those meetings. The idea was born from the perceived necessity of having a simple yet powerful platform for planning and managing events and being able to check for events that might be of interest to me in a specific location.

Having detected one niche that can be addressed with a product is the commencement of a product development strategy but far from being the end of the cycle. Developing a product for the web (in reduced terms, a website) requires serious thought and a well-defined plan, covering the technical details on how to develop the product and, equally important, how to market the product and turn it into something successful. No matter how good the product might be, if it’s improperly advertised and no one comes to know it, it won’t sell.
1 Introduction

Engineering as a Path to Business

While other areas of engineering like civil engineering appraise the entrepreneur and management side of their creations, those related to the computer science field usually relay it to others to do that job. It’s our understanding that there are a lot of opportunities being wasted by doing so. The work of an engineer doesn’t end when the product is technically finished and well conceived. If it’s not being used and valued by people, then the engineer’s work didn’t have any practical results.

Hence, the project hereby described and this thesis target not only the technical achievement of properly devising and accomplishing the successful development of a product but also its commercial availability. The description of the work is not only put in terms of technical excellence but also from the perspective of an useful product. Aiming for perfection is unattainable and, therefore, effort should be put in making something that users will be pleased in using.

It’s our belief that engineering and management/marketing can be coupled. Being so, connecting the engineering process to the management of the available resources and to the conception of a marketing campaign is a work of extreme usefulness to anyone contemplating investing in the sector.

1.1 Objectives

In the following work we pinpoint the perceived necessity of planning and advertising events and we present a proposal for addressing such issue. Furthermore, we devise a strategy in technical terms for the steady conception and development of a web product. Equally important is the emphasis put onto the ecosystem that surrounds the product and its competitors. More than the accomplishment of a well-outlined engineering process, the work hereby described must also consist of a good of value to others.

The product incorporates two key concepts: the one of a social network and the one of planning, managing and advertising events. Both concepts are tightly connected although they can exist and be used separately. We’ll explore and study how they can co-exist and why a social platform adds value to the offer.

To attain such product there’s an underlying engineering process that needs to be properly outlined and is covered on the following sections of this thesis. Typically, the process involved in conceiving a software product starts by selecting a development methodology; planning the development and testing phases as well as allocating time and other resources; selecting technologies; devising an architecture; developing a database to accommodate the business logic data; defining usability and user experience goals and formulate a consistent and effective testing plan. On web applications such as this one,
briefly looking upon a scalability strategy that would allow the website to accommodate increasing visits is prudent, in case of a sudden surge in popularity.

As previously mentioned, this proposal cannot be seen only from the perspective of a technical achievement. Its success is also dependent on how users value it. Therefore, management and marketing will play a major role in such work. Considering this we’ll try to analyze the current ecosystem of social networks and conceive a strategy for involving the community and making the website grow. Being technically superior to the competitors is not enough — taking a different course and figuring out a way to attract users to another social network is the main challenge to the product’s success.

In the end we’ll have a full blown web product that allows anyone to create, manage and advertise their own events in a simple way. All of this on top of a social network, allowing users to connect to each other, share their events, their photos and exchange messages. This will not come at once though — there are three separate development stages. This thesis covers up to the first stage where the product is privately tested, major bugs are tracked down and usability tests are prepared to evaluate the interface.

1.2 Motivation

Engineering is a discipline of great thought and method. Creating a product is an exercise of creativity and market perception. Engineering a great product is much more than the confluence of such virtues. It’s an escapade to the practical world, where theory distorts and meets practice. Add to this the commitment to market and sell the product and it’s a whole new adventure in the world of entrepreneurship.

The motivation behind the work hereby described comes largely from the author’s will of entering the world of entrepreneurship. Engineering has given him the knowledge on how to use and adapt tools to create, and the passion for economics, management and entrepreneurship provided the curiosity and courage to try. The outcome is the reflex of how valuable the product will be to people.

Furthermore, the possibility of building up a product from scratch, which not only requires a strict methodology but also the assembly of several assets ranging from self-discipline to work and perseverance.

Gathering all the pieces of theoretical thought and practical experience and combining them together into a full working project is a multi-disciplinary experience that enriches its creator and will eventually add value to the market.
1 Introduction

1.3 Social Networks

A social network is a social structure made of nodes (which are generally individuals or organizations) that are tied by one or more specific types of interdependency, such as values, visions, idea, financial exchange, friends, kinship, dislike, conflict, trade, web links, sexual relations, disease transmission (epidemiology), or airline routes [Wikipedia(2008)]. The analysis of social networks was first coined by J. A. Barnes as part of his work of studying a Norwegian fishing village and how their inhabitants interconnected and established social bridges [Barnes(1954)]. Figure 1.2 depicts how individuals typically can interact with each other and form social networks.

![Figure 1.2: An example of a social network diagram.](image)

Social networks have been rising in success in part due to the very humane necessity of people to socialize. Barriers to the socialization process like distance, timezones or even the unknowns once preventing the meet up are nothing but virtual on the Internet. In addition, the Internet provided the necessary ecosystem for people with similar likings to hang out together. Often virtually but sometimes resulting in successful live meetings.

This kind of social websites have therefore become a potential business. Although there’s no established way of making money on the web, several companies have been very successful exploiting the popularity of their products by the means of advertising.

The web is becoming social. Non-social websites still exist but they lack the interaction between users. By forfeiting on this very humane condition of promoting intercommunication between individuals they look less attractive than they were a couple of years ago. Even considering informational websites like online encyclopedias or newspapers, we’ve been witnessing the rise in popularity of the brands that have focused in collaborative
work and user interaction. Wikipedia, a collaborative encyclopedia written and validated by its users is a clear case of success of a network that promotes social interaction.

The surge and explosive growth of social websites like MySpace, Facebook or Hi5 gives a hint that the future is social. Based on this premise, it’s our top priority to provide a social platform that users can use to connect to each other. Strengthening the acquaintances by providing the tools to exchange messages or photos has evolved from being an extra functionality to a major miss when nonexistent.

1.4 Event Planning

People with similar interests often hook up together in an ordered or sometimes unordered fashion. Fans of electronic music can often be found around clubs in the same way that rock fans go to the concerts of their favorite bands. Spontaneous meetings might also happen by chance, inverting the traditional cycle of social contacts where two persons that know each other schedule a certain event. The key point is that people socialize and gather together.

The advent of the Internet leveraged how people connect to each other. Besides meeting live, individuals now have the chance to set up virtual meetings and connect with known or even unknown contacts. Social networks gave another boost to virtual acquaintances by making it easier to connect individuals with similar interests. Finding someone with interests similar to yours might take a couple of seconds (or perhaps hours, depending on how picky one might be). Common interests usually lead to the same events — rock fans will probably attend similar if not the same events. People gather together with friends, family or business contacts.

Planning an event includes budgeting, establishing dates and alternate dates, selecting and reserving the event site, acquiring permits, coordinating transportation and parking and, of most importance, inviting the attendees and letting people know about the event. Planning a dinner with a few friends is not such a hard task — pick a restaurant, schedule a time and invite friends over. This might get a little bit harder to manage if some of the friends are vegetarian or if they’d like to have dinner on a restaurant that allows smokers. Planning a conference expecting 500 attendees is clearly more complicated. Contacting the attendees and keeping them updated about upcoming news might be a challenge by itself — let alone pick a site, reserve a restaurant or select cheap hosting.

The Internet can be a useful tool for helping to plan these events. Occasionally it might be used to replace live meetings but is more frequently used as the medium to contact the attendees. Turning the event planning into a fun and simple process will certainly reduce the inertia implicit to start up something. It’s not the purpose of this proposal to replace live meetings with virtual meetings but rather to provide users with the tools to easily plan, manage and advertise their events in order to bring people together — in real life.


2 State of Art

When Charles Duell, Commissioner of the U.S. Patent Office, claimed that “everything that can be invented has been invented” he couldn’t foresee such upcoming revolution (it took almost a century though) as the Internet. But his quote isn’t that far from the truth when we take a close look at the Internet. Everything that’s created is not built from scratch but it is rather adapted, evolved and somehow inspired from previous work. If we interpret his sentence from a perspective of true, abstract creation in the inner context of the Internet it isn’t as non-sense as when applied to the real world.

The evolution of social websites wasn’t a revolution *per se*, at least in the same terms coined by Duell as inventions (like the creation of the automobiles). Social networks use the same technologies as any other website but add a concept that already exists in real world but had never been transposed to the virtual world: friendship. Social websites allow its users to connect to other users by declaring a friendship status.

Actually, even social websites have been subject to iterative evolution. There’s a huge gap between social websites of nowadays and those that first appeared more than ten years ago. They still share the same principles — allowing users to network between each other — but current websites collect much more information and use it in a much smarter way than of those built a decade ago.

But despite existing for more than ten years, social websites have been quite unknown until a couple of years ago. The sudden rise in the success of social websites has been mainly driven by the conception of new social features that were once possible but technically not feasible, like uploading several high quality photos (due to narrow-band). This led to the massive growth of these type of websites and to its success.
2 State of Art

2.1 The Social Web

The web didn’t start social. It started as a place to drop some ad-hoc documents and exquisite company pages that could perhaps be of some interest to others. Actually, a lot of people — especially private held companies — had serious doubts whether the Internet could ever be of any use or play any significant role in their business.

As technology started evolving at a faster pace and the Internet access becoming an increasingly faster commodity, the World Wide Web started shaping into a more dynamic form where users could now interact with each other. The web was becoming social. Users would group along with those sharing similar interests and would initiate on conversations with them. Not only web sites were becoming social, other web facilities like IRC (Internet Relay Chat) or IM (Instant Messaging) propelled the connections between people.

2.1.1 Social Networks

The first projects bringing forth social capabilities that appeared on the web were interestingly providing those features not as an end but rather as a means to a particular purpose. Classmates.com\textsuperscript{1} was first launched in 1995 and focused on ties with former schoolmates. This was a very particular social structure providing social tools not focusing on the network itself but rather on the concept of connectors. This niche of social networks is gaining traction again.

SixDegrees.com\textsuperscript{2} and Epinions.com\textsuperscript{3} were two other contenders, launching in 1997 and 1999 respectively. SixDegrees was more of a social platform than any other website until then. It’s intent was to connect any kind of people and allow users to disclose information about themselves. But according to its founder it was ahead of its time and by not having a way of generating money (web advertising was yet to take form) it had to close its doors. Epinions was more focused on trust based relationships, allowing people to comment and review products. The object of transaction of this particular social network was product reviews.

Evolution of social websites was slowly taking form but only in 2005 the revolution started happening. MySpace.com\textsuperscript{4}, a social website in its true form — allow people to connect to each other, define friends and groups, send messages, etc. — truly took this new kind of websites into a whole new dimension. MySpace soon outpaced Google in the number of visits and that caught the attention of Rupert Murdoch’s News Corporation

\textsuperscript{1}http://www.classmates.com
\textsuperscript{2}http://www.sixdegrees.com
\textsuperscript{3}http://www.epinions.com
\textsuperscript{4}http://www.myspace.com
whose decision to make one of the first biggest acquisitions in the virtual world helped launching this new and unexplored market.

Suddenly there was a whole new world for social websites to be and as with any other business, projects started showing up. Facebook.com\(^5\) was born out of Harvard Business School as a means to connect students but immediately became the place for people in general to virtually connect to each other. Facebook set another milestone in the Internet world by being one of the first websites to receive venture funding after the dotcom bubble.

Other websites like Hi5.com\(^6\), Friendster.com\(^7\), Orkut.com\(^8\), Hyves.nl\(^9\) and many, many more soon emerged and got its share of success. Hi5.com, a San Francisco-based social network website is the number one website in Portugal according to Alexa rankings [ale()] and is becoming hugely popular in many other latin countries, especially in South America.

### 2.1.2 Events Planning and Management

Websites permitting the planning, management and advertising of events predated the rise of social networks. Evite.com was launched in 1998 allowing people to easily create a page to advertise their event and invite people to attend it.

With the advent of social networks event planning websites started adapting to this new trend. Besides providing functionalities to plan, manage and advertise events, websites like Upcoming.org\(^10\), Going.com\(^11\), Socializr.com\(^12\) and many others also promoted networking between people. One can connect to friends, share photos of events, send messages or create common interests groups. Events are themselves a gathering of people with similar interests and transposing that to the virtual world has become a successful move.

The “nichedising” of the Internet (refer back to the Long Tail theory in section 1) and, more particularly, the “nichedising” of social networks translated in the surge of events websites focusing on particular types of events. Going.com is particularly suited for planning night events like clubbing. Last.fm is used for events related to music like concerts. This contributed, yet again, to a whole new market to explore.

---

\(^5\)http://www.facebook.com  
\(^6\)http://www.hi5.com  
\(^7\)http://www.friendster.com  
\(^8\)http://www.orkut.com  
\(^9\)http://www.hyves.nl  
\(^10\)http://www.upcoming.org  
\(^11\)http://www.going.com  
\(^12\)http://www.socialzr.com
2 State of Art

Notwithstanding this tendency, more general social websites have also been integrating events into their platforms. Facebook.com has a quite comprehensive platform for creating and sharing events with friends. MySpace.com also allows creating events and inviting friends. This can be easily explained if we analyze the revenue model of such websites but that’s outside the scope of this thesis.
2.2 Comparison

Each social website has its own particularities and its own competitive advantages. The Internet is such a massive compound of different users that it’s highly likely that certain functionalities no matter how exotic and apparently weird will strive. Twitter.com was born of such faith: even unthinkable functionality might gather the user’s interest. Twitter allows one to simply say what they’re doing but just in 140 words, lest the arbitrary propulsion of people to start writing lengthy articles. What initially seemed to be totally useless became a case of study for its sudden growth.

Furthermore, we’ve been witnessing from anecdotal evidence that functionality and technical excellence not always win. Technically inferior social networks like Orkut have been extremely successful in countries like in Brazil, mostly due to the fact that it had a properly translated version of the website to Portuguese Brazilian. They worked on fixing the gap in terms of features and overall quality of the application but that didn’t stop them from being successful and, finally, bought by Google.

This project doesn’t aim to be the holy grail of social networks or event websites. We want to build upon what’s currently available and make it a more enjoyable experience to the user. We want to offer further functionality learning from the patterns of success that others have been marking.

In order to fully assess what’s currently available on the market we present as an exhibit Table 2.2. It defines some features that have been crucial to social networks (and hence the fact of being common to most of the websites) and we mark whether the selected group of leading social networks/event websites implement them or not. At the rightmost column we show how the project stands compared to them. A detailed overview of each feature will be defined in further subsections.

2.2.1 A. Social Features

As Social Features we define the set of functionality required for a website to allow the constitution of social networks. Typically, the user has the ability to create a profile with personal details, add other users as friends, add photos and albums, comment on their profiles or photos, send messages, set a status (as in, what am I doing right now or the current mood). Additionally, some social networks make available the possibility of creating groups of similar interests, set up a blog or personalize the profile and public pages with custom themes. Privacy settings are given interchangeably as a good practice to protect the user’s privacy.

Our proposal is not far from the social features offered by other websites like Going.com, Facebook.com or MySpace.com. Evite.com has never built or intended to build its
Table 2.2: Table comparing the most important social websites currently operational.

The cross (x) denotes a feature that’s available and a dot (.) not available up to the time of writing.
infrastructure on top of a social platform but rather give the users the ability to create postcards with event invitations and hence the lack of social features. Other websites like Upcoming.org or Socializr.com provide ad-hoc functionality but they’re not a true social platform at their core.

We do not provide blogging though. We believe there’s a vast offer for blogging websites and it doesn’t add significant value to our proposal. Users can already send private messages and post public comments that might work as feedback to the event or to the person they’ve just came to know. All the event websites don’t provide it either, except for Going.com and Facebook.com. In the former, blogging is scarcely used by their users whereas in the second it’s used but definitely not as much as true blogging platforms like Blogger.com or Wordpress.com and it doesn’t add any perceptible value to the events platform.

We do contemplate the possibility of creating themes though, juxtaposing with the decision of leaving the blogging functionality out of the scope of the project. The reason for providing such functionality boils down to allowing users and providers to customize their pages so they can differentiate within themselves. Venues can add their own personal touch and create a custom profile with their logo or even their colours. Providers can be faithful to their brands.

All in all, the set of features offered by our proposal takes a good compromise between building a social platform but not diverting from the main goal of assembling the event management functionality into a seamlessly integrated package.

2.2.2 B. Events

Event planning, management and advertising constitutes the core of the project hereby described. Undertaking the mission of providing an events website requires providing at least a base set of functionality that allows one to create events, post them to the public (i.e., other users) and then manage the RSVPs. Expectably, all the sites covered in Table 2.2 offer this basic set of features.

Only MySpace, Socializr and Evite allow one to create themes for their events. It’s not a fundamental feature but similarly to profile themes, it gives some custom identity to the event. This is far from being a showstopper feature in what concerns to private events like a dinner but it allows one to differentiate from amongst the several hundreds (or thousands) of other similar events.

The point of divergence between our proposal and the other existent websites is the possibility of customization. Our approach to event customization is asserting that there are multiple categories and subcategories of events and each event has particularities of its own. When planning a soccer match between friends we give the possibility of
defining attributes such as the name of the teams, a maximum number of players per team, the type of field or how the teams are composed. Planning a dinner between friends differs completely from planning a soccer match and we therefore give a different set of customizable attributes. Information like the menu, whether attendees can smoke inside the restaurant, the maximum capacity of the venue or the availability of vegetarian meals does not make any sense in the realm of a soccer match but it’s a perfect match to what a dinner concerns. Furthermore, service providers are allowed and actively called for disclosing this kind of information so that users can know what they can count on and make their choices based on this. Choosing a venue for an event is then a matter of setting up their preferences and letting the system look for a place that matches the requirements. The search is not limited to the others home place but to any other city in the world that’s part of the application.

By contemplating the possibility of customizing events we believe our proposal outdoes what’s currently available on the market, with the exception of Facebook that also lets one customize venues and events, although to a lesser degree of customization.

2.2.3 C. Venues and Service Providers

The third point of comparison between existent projects is that of venues and the extent of functionality that’s made available.

Contrary to most of what’s currently available, we differentiate between a normal user and a service provider to the extent of permitting providers to define their own particularities in the same fashion that events are different amongst themselves. Derived from this, profile pages are also customized when information that’s particular to the service provider. A disk jockey will be able to describe the kind of music he plays and a restaurant the menu that’s up for selection. Facebook and Going also make this point clear and this distinction between users and providers, defining different profiles and asking for different information depending on the kind of provider.

In what concerns to searching for a particular provider our proposal goes a little bit further by allowing one to search for a venue based on the name and location but also using the attributes that have been previously defined upon the registration process or altered afterwards. As briefly touched upon on section 2.2.2, one can search for a restaurant that features vegetarian meals.

By binding this degree of customization to a search tool we empower users with a tool that will help them to find the venue they’re looking for. Naturally, the amount of service providers and venues that will be given as search results will depend on the popularity of the application.
2.2.4 D. Event Browsing

The social platform built underneath the application has a purpose: allowing users to network with other contacts but also making it easier to find events. Apart from planning their own events, users have the chance to find events they might like. We currently allow users to search for events in any country or city in the world; check the friends’ events, look for popular events and execute a search that can contain anything from the event description to its location or title. Also, a recommendation system based on [Lopes(2007)] is under the works for a later release.

As expected, these features can be found on most of the competitors listed in Table 2.2. Searching for events by city is available on websites wherein users can accurately inform about their physical location. Going partially implements this functionality as it allows searching for events by city but it’s limited to the cities they’re focused on — by the time of this writing only American cities. Facebook and our project as well is able to provide a more complete search as it covers approximately 272 thousand cities in the world.

Implementing popular events is one of the simplest functionalities such kind of websites can provide. As for friends’ events, it depends on whether there’s also a social network beneath the platform. If acquaintances can be virtually formed it’s then trivial to implement such feature as listing the events friends are attending. Upcoming and Evite lack this feature.

2.2.5 E. Localization

As it will be further explained in section 3.4.9, Localization consists of a more comprehensive approach to what is normally known as Internationalization.

Both Facebook and MySpace are translated into several languages and are open to registration from users based in any country in the world. This approach is contrary to thereof Going where they explicitly focus on a country (USA) and on its main cities like Boston, Chicago, San Francisco or New York. On our project we decided to take exactly the same approach as the two contenders previously noted as we believe it pays off the enormous effort of supporting multiple countries and multiple languages. By not limiting users to and from a certain location we open up a whole new set of possibilities.
3 Spateo - Conception and Development

As the basis of this work we propose a system capable of allowing users to create their events, edit information and customize it based on the type of event. Furthermore, users can manage all the event promotion and advertising. Invitees can RSVP (from the French “Répondez s’il vous plaît” — “Answer please”) to let the promoter know whether they’re attending the event, not attending or still unsure. Users can also leave their feedback on the event and post pictures. The event promoter can contact the attendees at any time to inform them about updates to the event. Relevant updates like a change in the venue are forwarded by the system to the attendees.

On top of the events platform, there are social enabling features to let users connect to each other. Users can create their own profiles, add other users as friends and create common groups of interest. In addition, users can post pictures and comments on other users’ profiles. Most importantly, users can check what events their friends are going to attend and let them know where they are.

Event promoters will be capable of creating and customizing their pages to advertise their events. Also, they’ll be able to connect with their attendees by letting them sign up to a custom newsletter that posts the promoter’s events.

In order to leverage how users consult the events near them the system tracks the location of the user. All events and users are set to be in a particular location and users will be able to check the events taking place in any city in the world.

All these features already exist, whether provided by a single website that combines a social base and an events platform or by the junction of two or three websites that separately mimic these features (for instance, some people create the event on Evite.com but use Facebook.com to advertise). The competitive advantage doesn’t come from implementing the exact same features in a better or simpler way but rather from providing extraordinary features typically not seen in these kind of websites.

Most of the social networks and event websites aforementioned in sections 2.1.1 and 2.1.2 allow connecting to users, sharing data among them and add and advertise events. But most of them treat events in the exact same way. Events differ from each other, though. Planning a conference is far from being similar to planning an indoor soccer match. Requisites, required information and even advertising is way different.
What we propose is a system that actively identifies these differences and provides the ability to customize an event depending on the type of event it is. For instance, while planning a soccer match we allow the user to add specific information about that match, namely the number of players per team, name of the teams or even the kind of soccer field. We also present a visual representation of the teams over a soccer field image. Attendees can select their team and that shows up visually. Planning a conference differs drastically. We’re not interested in the type of ground of the venue but rather how much attendees will have to spend in accommodation and other expenses, where can they spend the night, the agenda for the conference, panel of speakers, workshops, etc.. This can also be scrupulously detailed in our platform as we’ve previously defined what information each kind of event might require.

We also don’t lose focus on the social capabilities. Instead of moving from the events to the users, we propose a two-way relationship where users are also tempted to check and advertise their physical presence to their friends. By centering the attention on users and their relationships we’re offering a whole new possibility that has only been explored by web instant messengers like Twitter.com or Pownce.com but not specifically targeting events and physical location information but rather non-specific information.

Furthermore, our competitive advantage also comes from providing a localized version of the website to each user. Instead of following the typical approach of “a .com to rule them all”, we offer a different website (despite being technically the same) to each country.

### 3.1 Functional Requirements

All the functional requirements derive from the two chief properties of this project: an events platform where users can plan and promote their events; a social network platform that allows users to connect to each other and share information.

There are four main groups of functionalities that directly derive from the aforementioned two properties and then further down branch into other sub-functionalities. These are:

- Events Management
- Events Browsing
- Social Network
- User and Profile Management

The Events Management component is the core of the application. It allows users to create their events, search for venues that fit their requirements, advertise the events among friends and track the RSVPs.
The Events Browsing block is complementary to the Events Management component, as the former wouldn’t make sense with the latter. It provides a mechanism for searching and browsing events that might be of interest to the user. He can specify what kind of events to look for and where they’re taking place.

All the application sits on top of a Social Network. The Social Network allows users to connect with each other, share preferences, photos, create groups of similar interests and share with friends the events they’re attending. Furthermore, it’s also useful for advertising events and keeping track of those users that follow the promoter everywhere.

Finally, there’s the User and Profile Management component. All big applications, social or not, track user registrations for several purposes. In this case registration is not mandatory but unlocks new functionality. Registered users will be able to keep a profile page that will be used as part of the Social Network. A profile page is a place where users can expose their personal information like things they like, what they’re doing, photos, groups and events they might be interested in attending or even attending.

How all these components mash together and are used is better depicted in Figure 3.1. A common, unregistered user is capable of doing everything save creating and managing new events. Attendees have access to the event information and updates. Attendees consist of users that upon RSVPing to an event are promoted to this role. Finally, registered users that create events are named Organizers and are able to manage an event and contact Attendees.

All the functional requirements planned to be included for the first alpha release are thoroughly listed in the “Analysis and Specification of Functional and Non-Functional Requirements, and Test Policies” document in Annex A. Each requirement is listed in terms of priority, title, description and dependencies. Dependencies help planning the evolution of the architecture and priority helps planning the roadmap and define milestones. We’ll briefly touch upon use cases for each component in the subsequent sections.

### 3.1.1 Events Management

The events management functional module can be divided into several sub-modules. This module is the core of the application and should provide users with a system capable of adding, editing, removing and managing events. Moreover, the system must be capable of providing specific functionality depending on the type of event that’s being inserted. For instance, if the user is planning a dinner he must be able to specify not only the
location and venue but also what kind of food will be available, estimated cost for the dinner, etc.

Further functionalities shall also be available, like customized search of venues. The event organizer shall be able to search for venues that comply with his preferences, like searching for restaurants that have vegetarian meals on their menus.
Figure 3.2 shows a UML diagram of the possible use cases of the Events Management component from an Organizer actor perspective. For further details refer to Annex A.

### 3.1.2 Events Browsing

The system must provide with a way of consulting events depending on location, date and venue. Furthermore, the physical location of a user (country, city) shall be automatically determined or manually retrieved within the process of registration. Based on this information, the system shall present a listing of events happening on that area or on nearby areas in case there’s none on that particular area.
Figure 3.3 lists the use cases of the Event Browsing component. For further details refer to Annex A.

### 3.1.3 Social Network

The application should provide the typical functionality of a social network, namely being able to create circle of friends, send messages within users, create common interest groups, share pictures, etc. Furthermore, the social network shall be used as the means to further extend the events functionality by allowing users to advertise their events among friends, post them to groups, etc..
Figure 3.4 is a use case diagram of the Social Network component. Most fundamental features and how they correlate to each other are listed. For further details refer to Annex A.
3.1.4 User and Profile Management

The system shall provide a user registration form, a user login and logout form and an access list to filter which users can access which pages. For instance, events ought to be created only by duly registered and authenticated users. Failure to do so must carry with an invitation to the user to proceed with the registration on the system.

Furthermore, the user must be able to customize its profile by inserting his or her own personal information, photo pictures, status, profile comments, etc.

Figure 3.5: User and Profile Management use case diagram.

Figure 3.5 shows use cases for the features expected in a user and profile management system. For further details refer to Annex A.
3.2 Non-Functional Requirements

Non-functional requirements refer to all the properties that can’t be materialized in terms of features but are still part of the application. These consist of:

- Usability and User Experience
- Web Standards
- Performance and Optimization
- Internationalization and Localization
- Security and User Privacy

These requirements are not tangible to the user but are of utmost importance to guarantee that the application is performing correctly. Usability and user experience define good principles on how the interface should be conceived, similarly to how design patterns suggest good solutions to common software engineering problems.

Web standards open the way to having an API in the future and letting the more tech-savvy users make use of our data to create third-party applications. Web standards also define norms of how web documents should look and be formed.

Performance and optimization goals are crucial to sustaining bursts of growth without reducing the application’s availability. They can’t deal with an explosive growth in visits but can leverage sustainable growth.

Internationalization and localization brings the application closer to the user by presenting it in a way that’s familiar to his culture and country.

Finally, security and user privacy guarantees that the user’s private information is safe with us and that other users don’t sneak in to access information they’re not supposed to.

All the non-functional requirements planned to be included for the first alpha release are listed in the “Analysis and Specification of Functional and Non-Functional Requirements, and Test Policies” document in Annex A.

3.3 Timeline and Expected Results

There are four separate phases for the project release. The first one marks the end of the implementation of all the functional and non-functional features defined along sections 3.1 and 3.2. This milestone is called “feature-freeze” and is the commencement of the
alpha stage. The alpha stage release will be only open to a selected group of people that will be responsible for testing the application. This group of people will be individually contacted in order to be warned that data loss may occur and they may step against (and probably will) application bugs, some fatal and others mere nuisances. Also, another group of individuals will be subject to usability tests to assure that the application is being correctly understood and used.

During the alpha stage, the development team will be collecting all the bugs, user suggestions, performance issues and usability tests. No new features will be implemented even per users’ suggestion. The goal is to correct all the detectable issues. Notwithstanding this procedure, all the user suggestions will be taken into account for a future release.

Whenever all the traced bugs are corrected, the application will then be open to the general public. This initiates the beta stage. By this time, all Internet users will be able to access and try the application. A beta release is still considered to be unstable, meaning that data loss and other issues may still occur. The beta release is typically more stable and mature than the alpha release but one might still encounter bugs (although the Web 2.0 trend has brought us products that never leave beta stage despite its evident stability, like Google’s Gmail).

There are no plans for defining a roadmap to when the beta stage is due to end. It will depend on the number of bugs that emerge and on the time required to properly translate the application to an initial set of key languages.

With the end of the beta stage we’ll have the official launch of the product in several countries, mostly in Europe and the United States. This release should be stable and no major hiccups found. This does not prevent though technical difficulties from arising due to unexpected growth. Several websites have been subject to the less positive consequences of popularity and that required more time and money to solve.

![Development and product launch roadmap.](image)

Figure 3.6 depicts the expected product development roadmap up to the launch day.

In terms of user adoption and expected results for growth, it’s expected a stall growth during the alpha stage considering that the application will be closed to the general
public. Upon the beta release, it’s likely that an initial user base consisting of a very
devoted group of users will start using the application and helping to build a community.
It’s of paramount importance to provide feedback mechanisms to these users and also
make sure every single mail sent by them is properly responded to. The gratification these
users are expecting boils down to the feeling that they’re helping and being considered,
but they call for nothing less than that.

As the application grows, it’s likely to see a moderate to above average growth. Typically,
applications that enjoy a huge surge in popularity are a drastically new concept that
persists or immediately dies. Such a project like this very one is far from being a whole
new idea but rather an iterative work over what currently exists. Popular websites like
Facebook.com or Hi5.com took a long time to take off to Internet stardom.

3.4 Development

3.4.1 Planning

Planning the development of an Internet application is a very time-consuming task. Not
only one has to plan in terms of functional and non-functional requirements, it’s also
necessary to outline the development phases, dependencies, goals, measurable results
and dependability.

The planning of this project doesn’t follow the conventional route to planning: sketching
precisely what to do and then proceed with actually doing it. The approach was less
ordinary and much more flexible. The underlying idea of the whole project was soon
set forth, but the listing of functionalities is still an undergoing process. The project
is continuously adapting to the upcoming technology trends and that can’t simply be
planned in detail as compared to conceiving a car, where every single detail has to be
exhaustively thought of.

In order to fully jolt down the functionality from a macro-level perspective, a top-down
approach was taken to plan and devise the project. By thinking in terms of feature-
wise we focus on the user needs rather than technical details and liabilities that usually
divert the programmers. Crucial aspects of the application are summoned first. These
are the very nature of the project — an events website where users can plan, manage
and advertise their events — and the social network beneath it that will be handling
the acquaintances between users and provide the necessary tools for users to properly
connect to each other.

The amount of features related to those two main requirements will then depend on
the undergoing study of the project as a business model, on the constantly changing
Internet trends and on the time available to implement them. Most of the functional
requirements devised for the initial release are listed in section 3.1, but this list is far from being finished and is constantly evolving and subject to change. For instance, when the project was started no plans were made in relation to supporting a mobile version of the website due to the fact that this project requires a rich Internet experience that mobile phones couldn’t provide up to then. With the advent of the Apple phone the support of a mobile version is now being seriously pondered. How things will evolve will be conditional on how the mobile phone panorama will advance.

Thinking up of the functional requirements accounts for only half of the planning. It serves no use an application rich in features but lacking an accessible interface that users can use. Or an application that’s constantly down and not ready to scale or deal with performance issues. Therefore, other non-functional problems have to be anticipated and a plan drawn to address them. Recovering from a serious technical problem can depend on previous anticipation of growth. Some non-functional requirements were also defined in section 3.2. Despite this serious effort in trying to predict performance and scalability issues or usability requirements, some situations are simply not controllable (like the server burning to flames) but this pays a serious advantage when compared to no planning at all. Again, these are subject to change as they must adapt to functional requirements and architecture changes.

Overall, the planning was done at a macro-level, skipping the detailed 500 pages documents that go as far as writing method signatures. Having a structured list of requirements (functional and non-functional) is a handy tool to focus on the problem in hands and avoid solving imaginary problems that engineers usually come up with. This list pairs perfectly with the development methodology adopted, referred in the next section. Despite skimming over the traditional lengthy panoply to planning, this process is still done and of extreme importance to the proper conception of the product.

### 3.4.2 Development Methodology

Back in the days, software used to be developed in a very sequential and well-defined way. Requirements were gathered from meetings with the client, analysis and design was done and planned within the analysis team. Soon after, the development team would succeed to execute the development and whenever it would finish, the quality assurance team would be encharged of verification and validation. Finally, the product would be handed to the client and maintenance would be eventually done whenever necessary.

This classical approach to development (in particular, the Waterfall model [Royce(1970)]) was first put to use by the Rolls Royce car manufacturer and then adapted to the development of complex and big software systems. But software is not done and assembled in a way similar to cars. Cars are projected and its design stays frozen for a couple of years. Also, requirements don’t change after the car is built and the client doesn’t change his mind about how the front panel should look like.
Most of the software development is more of an art, chaotic by nature. Requirements are constantly changing and so is design and architecture. Multi-client applications (like websites) demand architectural adjustments to accommodate scalability, as soon as they grow in popularity and that might mean complex changes. Furthermore, clients are constantly changing their minds about the feature set that was defined just a week ago.

So, in order to cope with this unorganized methodology inherent to this project we require agility. We need a development methodology that is adapted to the client, integrates him into the development process and allows subsequent changes to the functionality and the overall design of the application. Software is not stationary and the development process should work hand in hand with that fact.

Considering the monolithic nature of the project — the client is at the same time the developer, engineer, tester and architect —, that could make it easier to use a classical development methodology. But the author’s mindset of the project in its beginning and during the development has changed substantially. New features have been concocted while others have been removed. The underlying architecture had to adapt to this. Therefore, in its essence, the project is still very dynamic and flexible and an agile methodology is the best approach to such requisites.

Therefore, by choosing an agile approach, we are in concordance with the Agile Manifesto [agi()] that states the following:

- Customer satisfaction by rapid, continuous delivery of useful software
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Even late changes in requirements are welcomed
- Close, daily cooperation between business people and developers
- Face-to-face conversation is the best form of communication (Co-location)
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
- Simplicity
- Self-organizing teams
- Regular adaptation to changing circumstances

3.4.3 Architecture

Web applications typically share a three-tier architecture, with the data layer on the bottom, business logic sitting in between and the presentation layer on top of the stack.
This clear separation makes for an easier maintenance of the application since the logic tier is not split up among the presentational layer or in the database. On the other hand, the denormalization of this architecture by putting some of the business logic on the data tier (in the form of stored procedures, for instance) might speed up the application. Considering our long-term goals of maintainability we clearly opt for making the clear separation of these logical entities even if it comes up with a performance penalty as the application scales.

Figure 3.7: Typical three-tier architecture for a web application [wik(2008)].

Figure 3.7 visually depicts a multitier architecture along with a brief explanation of the data flow. As one can see, the typical workflow starts with the user inputting user data through an interface. On web applications this means doing an HTTP GET request to a specific URL or submitting data through an HTTP POST request. Data can also be submitted over other protocols but the most common one is HTTP.

The input data is then submitted to the logic layer that will be processing it. This typically means parsing the information, executing some additional tasks and creating a representational form that the storage layer can handle. Two kind of requests arrive to the data tier. Read or write requests. Read requests are usually faster to execute and return some kind of result. Write requests can take longer but only return a confirmation that the write operation succeeded.
The flow is then inverted and information goes from the bottom back to the user. The application logic layer might intervene again on the data coming from the data tier (and send it back again to the data tier if necessary) or simply send it to the presentational tier that will be formatting it, generating HTML and presenting it to the user.

The architecture of our application, although heavily based on this three-tier layer pattern, goes a little further as it requires more services to function. Besides the database, the module handling the business logic and the presentation view, we also manipulate data over the Internet, perform full text searches on a specific engine and pass messages for deferred tasks, like sending emails through an SMTP dispatcher. The physical architecture is depicted in Figure 3.8.

![Physical architecture of Spateo](image)

Figure 3.8: Physical architecture of Spateo.

The architecture of a web application gets more complex and detailed when we use an application framework. An overview of the used technologies will be made on section 3.4.5 but devising an architecture requires studying the framework architecture. Given the fact that we’ll be using Ruby on Rails as the underpinning to this project we’ll be able to build the application over its structure. Frameworks impose a certain architecture onto the application developers but gratifies them in return with its powerfulness and flexibility. The underlying architecture of the current version of Ruby on Rails can be seen on Figure 3.9.
As one can see, the Ruby on Rails application framework adds a couple more layers to the typical three-tier web application. For instance, ActiveRecord is put between the data and the business logic layers. Although it might seem to add complexity to the overall application structure, ActiveRecord executes object-relational mapping between data and structures, sparing the developer from writing common SQL queries and mapping them to class objects.

The whole application sits over Rails’ internal architecture. This is more beneficial than deleterious despite making the overall architecture look more complex.

### 3.4.4 Design Patterns

Design Patterns are commonly known as micro-architectures and hence the placement of this section after the Architecture section. These patterns are general reusable solutions
The most prominent design pattern used in the project comes from the choice of using Ruby on Rails. Ruby on Rails relies on the Model View Controller (MVC) pattern. The implementation of this design pattern helps isolating the visual appearance of the application from business rules, sometimes mixed-in if not properly accounted for. Actually, this design pattern is so important to the overall application architecture that labeling it of micro-architecture is not rigorous. Figure 3.10 depicts the MVC pattern.

![Figure 3.10: The Model View Controller object diagram.](image)

According to this pattern, controllers will be receiving and distributing input data and reacting to events. They usually hand it from user actions to the models or to the views. Views also have access to model information and are called by the controller.

So, transposing this to the web environment, we’ll have an event triggering a specific controller. This usually happens upon an user action (like submitting a form or simply requesting a page) and the selection of the controller to handle the request is done by parsing the URL. The controller evaluates the request and acts accordingly. This can be simply returning a view or, a little bit more complex, manipulate a model. Models are the object-relational entities of the database. Afterwards, the controller picks the correct view and returns the information back to the user.

For a more concrete example we can take a look at Figure 3.11.

In this case, the URL encapsulates all the information required for the controller to act upon. `users` tells which controller should be handling the request (considering we have different controllers for different set of actions), and what to do. So, the `users` controller will be called and be asked to show information about user 1. The controller will then query the `User` model containing information about users and ask it for the data related to user 1. The model returns the information about the user (if user 1 actually exists)
3 Spateo - Conception and Development

![Diagram of Model View Controller pattern]

Figure 3.11: A concrete example of the Model View Controller pattern.

and the controller will then pass it over to the show view to use it. The view picks the information it needs and generates the output code — what’s shown to the client.

Notwithstanding the MVC being the most prominent pattern in the Ruby on Rails framework, other patterns do exist within the framework. Chain of Responsibility — Chain the receiving objects and pass the request along the chain until an object handles it [Erich Gamma(1994)]; Iterator — Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation [Erich Gamma(1994)]; Template Method — Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm’s structure [Erich Gamma(1994)]; Adapter — Convert the interface of a class into another interface the clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces [Erich Gamma(1994)]; and many more.

Moreover, some of these patterns were ways to address shortcomings of some programming languages, mainly Java. Dynamic programming languages like Python or Ruby overcome many of these common problems (like providing built-in Iterators or mix-in capabilities to extend an object/class). Other patterns (like Adapter) are used to provide seamless functionality like connecting to different databases without having to change any line of code. Nonetheless, they’re proved solutions and they were therefore more of candid approaches than static and strict micro-architectures in the way that they used to be put to use in the past.

3.4.5 Technologies

In order to develop this project a set of technologies was chosen based on some important factors.
Open Source  Priority was given to Open Source technologies. By being Open Source we can ensure that we’re not dependent on a company that fully controls the software. Furthermore, by relying on Open Source we can manipulate the software to satisfy the project’s unique needs or even to correct critical bugs.

Cost  The software ought not to be very expensive due to the fact that there is no venture backing the project up to this date. Fortunately, most of the Open Source software is also monetarily free, which automatically complies with this restriction.

RAD  Rapid Application Development. The project was built from scratch by one single person, meaning that there were not several engineers to cope with Java J2EE frameworks’ endless XML configuration files or .NET obnoxiousness. Therefore, technologies ought to be flexible, interoperable, easily deployable and don’t get into the programmer’s way.

Documentation  All the chosen technologies have to have comprehensive documentation to assist in the development process.

Also, consideration has to be done to the fact that the application will be potentially handling several gigabytes of storage. Self managing a storage mainframe is not feasible in terms of resources (money and time) so alternatives for flexible and online storage should be looked upon.

Based on these constraints, the following software was chosen to address the project’s immediate needs:

3.4.5.1 Database

To serve our relational data storage needs we’ll be using the latest stable version of MySQL 5.0 with InnoDB as the storage engine. MySQL works in most of the operating systems (UNIX/Linux, Mac OS X, Windows, etc.), has a small memory footprint and makes efficient use of the CPU power. Also, MySQL is very fast, easily deployable and maintenance is also easily done. By choosing InnoDB as the storage engine we have ACID transactions, cascade updates, row level locking and several other niceties usually found on high cost databases like Oracle or SQL Server. Finally, high-load websites like Facebook.com, YouTube.com or Digg.com use it, serving as a full-proof that it scales up to thousands of requests per second.

Other options were also available. On the commercial side we have IBM’s DB2, Microsoft’s SQL Server or Oracle’s flagship product. They’re technically good alternatives but they require a significant budget that’s not available for this project. Therefore, none of them was considered.

On the Open Source side there are also other valid options besides MySQL. PostgreSQL or Firebird are powerful, reliable and fast databases to be considered. In terms of
requirements there were no particulars that would require one database instead of the other so the choice was done by picking the one that the team of developers was most comfortable working with.

Also, choosing MySQL does not add any kind of dependability as it can be easily swapped (due to Ruby on Rails model abstraction) and provides powerful tools for achieving a good scalability.

### 3.4.5.2 Full Text Search

Although some of the relational databases support full text search, most of them were not built for serving this purpose and are not as optimized as a full text indexer. Instead of relying on the database, we took the decision of using a dedicated full text search engine. To perform this job we chose the latest stable release of Sphinx, 1.9.7. Sphinx is an extremely fast and reliable full text search engine that can be easily attached to MySQL. It’s also very customizable, at least to satisfy most common needs.

Optionally, one could use the well popular Apache Lucene full text search engine. Lucene is a Java-based engine that’s well tested, optimized and very fast. Due to its dependency on Java it requires more powerful hardware than the C++-based Sphinx. Also, there are good tools for integrating Ruby on Rails with Sphinx. Despite being technically feasible to bind Rails and Lucene, it’s a task of its own that diverts from the project’s main scope. Some bindings exist but they’re still under development.

There was also a third option to be considered. Ferret is a Lucene-like full text search engine implemented in Ruby. Unfortunately, tests have shown that its performance is deceptive and far from being ready to be used on a production environment. Considering the potential growth of the application this option was immediately discarded.

Ultimately, we could have used PostgreSQL instead of MySQL. PostgreSQL comes with tsearch2, a very efficient full text search engine. This decision would come at a price though: it would increase the burden the database would have to deal with. Despite its importance, search is not as important as direct access to the database information so we can’t run the risk of queuing read requests while full text search is being executed.

### 3.4.5.3 Storage

As previously mentioned the application will be handling several gigabytes of information, most part due to photo uploads. There are several online services that allow storing and accessing files in an easily attainable way, but some of them are considerably expensive. Amazon S3 provides this online storage platform for very good prices. It also has a very good and extensive API allowing easy manipulation of the files.
Amazon S3 was actually the pioneer in offering an online storage web service and competitors are scarce. Those that do exist can’t afford to charge such commodity prices as Amazon does so the decision was immediate.

Alternatively, data could be stored locally. Up to some point (around 500GB) it would eventually be cheaper than using S3 but as soon as that barrier is surpassed prices usually skyrocket. Also, Amazon is responsible for guaranteeing the service availability, manage the hardware infrastructure and scale it, so that’s one less concern for our engineering team.

3.4.5.4 Web Development Framework

Developing such a complex application can be leveraged if proper tools are used to assist in such task. Picking a powerful and complete web framework will cut in the time needed to reinvent the wheel and rewrite tools that are frequently used by many — database access layers, security layers, helpers, etc. But the underlying language of a web framework should also be powerful and simple to use. Ruby on Rails satisfies both of these necessities. Ruby is a dynamic programming language in the likes of Python with a very clean syntax. Rails is a web framework built on top of Ruby that speeds up the development by seamlessly bundling together tools that are frequently required to construct these type of applications. It comes with an ORM (Object-Relational Mapping) library that automatically maps database tables to objects (similar to Hibernate but much more simple to operate): provides multiple helpers to ease on typical tasks like converting new line feeds to breaks; clearly separates the logic of the application from the interface by the means of the Model-View-Controller pattern, etc.. Much of its powerfulness is granted by Ruby due to its very dynamic nature. Rails version 1.2.6 will be used for development but an upgrade to version 2.1 will be made upon beta release.

Following the release and success of Ruby on Rails several other viable choices emerged. Django is another web framework built on top of the Python programming language that mimics most of the functionalities found in Rails. Symfony, TurboGears or CakePHP are other three MVC frameworks written in PHP that also try to follow the Rails approach to web development. Apache Struts and Apache Tapestry are other two well-known Java frameworks that assist the task of web development.

The choice was wide and considering that most of them offer most if not all of the same funcionality expected from an MVC web framework, it boils down to a matter of personal preference. Having chosen Django was also valid and would not impede the development of the project. Same could be said of using no framework at all and relying on pure language tinkering through CGI requests. The only constraint was the time. Web frameworks give an undeniable boost in productivity on very large projects.
3.4.5.5 Web Server

To serve the static files and routing the dynamic requests we need a powerful yet simple web server. We’re currently evaluating the newcomer Nginx due to its incredibly small memory footprint (~10MB per thread). For production deployment we’re also considering Apache. Despite taking considerably more memory per process spawn, Apache is more feature-rich and has a better documentation than Nginx.

Another alternative would be to use Lighttpd, a fast and lightweight web server. Unfortunately, several memory leaks have been recently found in the latest version, which compromises it for production use.

The web server is not, under the current configuration, a crucial piece of the architecture. Actually, it could be replaced solely by the application server at the price of slower performance (the application server is not intended to serve static files as fast as a pure web server). Also, there isn’t a technical requirement that requires the use of one instead of another. Therefore, the choice of using a combo of Apache and nginx was done solely based on the past experience with Apache and nginx’s excellent memory footprint.

3.4.5.6 Application Server

Ruby on Rails needs to be processed by the Ruby interpreter. In order to route and handle these requests we need an application server. The most common option for Ruby frameworks is Mongrel. But this is a constantly evolving field and other options like Eventd Mongrel, Thin or mod_rails have been emerging and might become relevant choices in the next couple of months. Therefore, we’re currently using Mongrel 1.1.4 to serve Ruby but this is far from being a definite and static choice.

3.4.5.7 Operating System

Most of the UNIX-based operating systems will fully integrate with the selected set of technologies. Linux, a BSD flavor or Solaris could be used but the author is more experienced with Linux. Developing was done on Mac OS X and the first phase of deployment will be made on Linux but, again, this might change depending on how the application evolves and the hardware needed to accommodate it. As soon as bottlenecks are factored out the operating system might be changed to take the best out of the hardware.

The software stack chosen also runs over Microsoft Windows, but Windows is costly and it’s, arguably, not as optimized for running these applications as a UNIX pure or derivative like BSD or Linux. Also, experience with the tools was taken in account so the usage of a UNIX-based operating system was straightforward.
3.4.6 Data Storage

Typical web applications have a medium for persisting data and user information. Most of these applications also require relational information to be stored and for this purpose relational databases convey the required functionalities to persist and relate data.

This project is no exception to this need of storing data and, as previously mentioned in section 3.4.5, a database is used to gather information. More precisely, MySQL, a lightweight and extremely fast database, is used to vow this task. The database is thoroughly used to store user information (like name, username, email, etc..), event data (all the data related to events like start date, end date, venue, location, etc..) and relational data like who’s friend of who.

Furthermore, the application allows users to add their own data like photos. Photos could be stored as BLOBs (Binary Large Objects) in the database, making it easier to converge all the storage mediums into one single tool: the database. But MySQL and most of the relational databases are not supposed to be handling photos or any other large files. Therefore, it comes at a price: performance. Moreover, databases can easily corrupt and rebuilding a storage engine can take hours depending on the amount of information stored. Doing the same process with several thousands of pictures stored might even not be feasible due to the time involved in completing such task. Also, storage is not free and getting a database capable of serving huge loads of data might become very expensive.

Assuming this, another medium for storing files had to be found. Files are usually kept on filesystem and most of the databases actually work over a filesystem (MySQL and others allow raw access to hard drives blocks but that’s outside the scope of this thesis) so that could be an option and it clearly is. But file storage and bandwidth might become expensive when passing around several gigabytes of information. Also, maintaining such a system is a demanding task by itself, clearly outside the reach of a single person.

Fortunately, Amazon.com realized exactly the same — engineers don’t want to worry about managing mainframes or clusters, they want their systems running at the lowest price possible. Amazon S3 is a service they make available for storing files and serving them. Prices are also very competitive and depend on the volume of information transmitted. Also, they assure all the maintenance, energy and whatever is required to have the system running 24/7.

Integrating Amazon S3 in the project was actually very straightforward. The workflow is initiated when a user uploads a photo (currently, photos are the only thing that users are able to upload but that might change in the future with the addition of new functionalities). The meta information about the photo like size, dimensions, date uploaded and filename are stored locally on the database of the application server for convenience. Photos are then uploaded to the Amazon S3 service and additional tasks (like creating thumbnails) are then performed.
Whenever a file is requested, the filename is read from the database and a query through Amazon’s S3 API (REST, SOAP, HTTP or BitTorrent) is done to retrieve the file and present it to the user. The user contacts directly with Amazon’s S3 cluster thereby reducing the connections and data flow required to pass the file around.

3.4.7 Testing

Testing is a fundamental way to assure that the application is doing what’s supposed to do (verification) and it’s also doing it correctly (validation). Furthermore, testing also ensures that previous functionality doesn’t break when new functionalities are added to the application (regression testing). Assessing that the application is behaving as expected is part of the Quality Assurance process and it’s a fundamental step in the software development activity.

Web applications have several dimensions of testing. Typically, the testing of web applications is potentially compounded of tests to the user interface; tests to the business logic layer and tests to the data access interface. More complex applications like those connecting to several services and disclosing APIs to the end users might require an extra layer of testing to ensure all the connectivity is occurring as expected.

In order to test these distinct layers of a web application different testing methodologies are to be used. Unit testing ensures that individual units of source code are working correctly. They target a small fraction of the source code, isolating it from the rest. Unit testing is of quintessential importance to testing data models and other sections of the application that can be easily defined in terms of input arguments, actions and expected output values. For instance, suppose a unit test that ensures that a currency is properly converted to another currency. The input is one amount and one currency, the action might be connecting to a web service that executes currency conversion or executing the conversion itself and the output is the value properly converted. The test passes when the output is the correct conversion of the input value.

Other sections of a web application, namely the user interface, require other approaches to testing. Defining an user action in terms of input, actions and output is harder and usually a very fuzzy task. Functional testing addresses this issue by taking a more high level and granular approach. Instead of expecting a number and a currency, functional testing evaluates the application in terms of requests and responses. When a form is submitted via HTTP POST the application typically redirects to a new page upon processing the form or bounces back to the form page if errors have occurred. Testing can be done by injecting errors to the form and asserting that the application is bouncing back and verifying that a correctly filled up form causes a redirection to occur to the proper landing page. Functional testing is not suited to assessing conversions are being done properly but it’s a very convenient way of testing the application in terms of blocks of functionality.
Finally, there’s a third testing technique that’s thoroughly used for assessing the application’s correctness. Unit testing is highly technical while functional testing makes it more granular by operating in the realms of functionality and expected actions. Behavior testing takes it to the next level by allowing the testing methodology to be defined in a natural language, combining it with the requirements specification. This has several advantages, namely helping uncovering some specifics about a particular feature that might not be very clear from the start; bringing the client in to assist in the planning, development and testing process; and defining the code in terms of added value and not simply technical functionality. This also promotes Test Driven Development, a development methodology where tests are written first.

By combining these three testing methodologies we bring together a very powerful and complete testing framework to assist in the Quality Assurance process. But testing is a discipline and requires a balance between time and necessity. The project is made of several thousands of lines of code and its functionality, although critical to the project as a business opportunity, is not as critical as life-dependable systems. Also, it’s not practical in terms of resources (time and money) to have automated tests to every single functionality of a web application. Therefore, critical sections of the application were elected to be properly tested as they guarantee the functioning of root features. More ordinary actions like checking if user interests or favorite musics are being correctly saved is only subject to manual validation. This decision comes at a price: previous functionalities might start malfunctioning and there’s no automated mechanism to check on them. On the other hand, it’s impractical for a single person to develop a whole battery of tests to assert every single line of the application.

The guides that were developed to conduct and assist on testing and to assess the application is working correctly are defined in Section/Annex A.

### 3.4.8 Usability and User Experience

Of no use is an application that has a large amount of functionalities, is extremely well devised and conceived but makes it hard for its users to actually use it. The frontend — what the users do see — is of as much importance to the overall feeling of the application as any other functional requirements like making sure code from the business logic layer is decoupled from the data storage layer.

The Usability and User Experience is usually seen as non-functional requirements but they should still be listed for designers and developers to bare in mind while developing the application. Usability is usually defined in terms of five quality components [Nielsen(1994)]:

**Learnability** How easy is it for users to accomplish basic tasks the first time they encounter the design?
3 Spateo - Conception and Development

**Efficiency** Once users have learned the design, how quickly can they perform tasks?

**Memorability** When users return to the design after a period of not using it, how easily can they reestablish proficiency?

**Errors** How many errors do users make, how severe are these errors, and how easily can they recover from the errors?

**Satisfaction** How pleasant is it to use the design?

These five dimensions of the usability of an application also reflect the whole user experience. User experience design is a subset of the field of experience design which pertains to the creation of the architecture and interaction models which impact a user’s perception of a device or system. The scope of the field is directed at affecting "all aspects of the user’s interaction with the product: how it is perceived, learned, and used." [Norman(1999)].

Usability plays a major role on social applications built out of user data. More content-oriented websites like online journals might survive without a good user experience as long as they present interesting content to the user. But social websites are built out of the user actions so user experience must be taken into serious consideration.

**3.4.9 Localization (l13n) and Internationalization (i18n)**

The Internet is accessible by millions of people spread around the world, sharing different cultures and talking different languages. If we consider only the English-speaking attendance we’ll be ignoring approximately 50% of the active population using the Internet. This will tend to increase even more as the internet turns to become a commodity in the emergent countries like China or India.

Therefore, it’s of subliminal importance to provide a localized version of the website for several countries. This potentially increases the user base of the application. Internationalization (commonly known as i18n) solved part of this issue by providing a platform for translating the text into other languages. But properly constructing an application that fits not only language requirements but also social and cultural aspects is a much more vast task than simply translating text. Localization (l13n) takes into consideration social, cultural and economical properties of a certain country, like the currency used or the metric system. To a further extent, localization might also mean censoring the application to fit certain countries’ requirements, like the Chinese regulations to the freedom of speech.

There are two degrees of localization supported in the application. First, the support to the translation of the application to any language, but especially crafted to target key languages like Portuguese, French, German or Chinese. Secondly, the application has a comprehensive database of all the countries, regions and cities in the world (totalling
about 252k cities) which allows grouping events and users by country, region or city. This geographical localization brings users together, leveraging the social connections between them.

In order to support the translation of the application the Globalize plugin was used. Globalize [rai()] is a Ruby on Rails plugin that allows the translation of model data (namely data inserted in the database) and view data. It also makes available a currency framework for converting currencies.

For locating the users and bind them together in terms of georeferential information two distinct approaches were used. The first one asks the user for the city and country upon registration. The second method resorts to a GeoIP service that traces the IP address and returns the country. MaxMind.com provides this service.

### 3.5 Generalization

Spateo is intended to provide support to several dozens of types of events. Each kind of event, as previously mentioned, has its own particularities. A dinner requires information about the restaurant, the menu, the availability of vegetarian meals, etc. Conversely, a soccer match requires information about the teams, number of players, color of the equipement for each team, etc. Each type of event has its own domain specific information, which has to be stored for further use.

From a technical perspective, this implies that the system must be able to distinctly manage and operate this data. In order to correctly store the data, it’s crucial to maintain the relational meta-information between events and their properties but it’s also equally important to keep the system flexible enough so that adding, removing and changing event properties and event types continues to be a straightforward task. In order to keep the database schema as simple and adaptable as possible it’s necessary to abstract and separate three chief concepts: events, event types and event properties. Alternatively, one could maintain a table for each event type, keeping the definition of event, event type and event property coupled together. This would likely increase the overall complexity of the database as one table would be required for each type of event. Hence, it should be avoided at the risk of turning the system into a tangled mess. Figure 3.12 depicts how the aforementioned concept would reflect in terms of the database object model.
Figure 3.12: A database example schema for storing information related to the events.

As can be seen from Figure 3.12, supporting $N$ different types of events would require $N$ database tables. Furthermore, properties used by more than one type of event (like the definition of *team* that’s throughout used in most of the events related to sports) would not be abstracted and inherited, having to be duplicated throughout the tables using them. This particular issue could be circumvented by creating self-referenced tables, solving the problem at the price of adding even more complexity.

The second problem derives from the first. It’s necessary to use and manage this information in the business logic layer. Object-oriented languages provide mechanisms to facilitate this. Using an inherited structure of classes to replicate the hierarchy of the event types makes it possible to reuse methods and properties. Additionally, the concept of inheritance avoids the problem of shared properties altogether. Still, class inheritance does not provide all the flexibility required. The application can’t be dependent on particulars like properties of events or demand an overall refactoring in order to add or delete an attribute. Figure 3.13 shows a sample of the class inheritance applied over event type classes.
Despite leveraging the problem, inheritance does not fully untangle it. First, the system must not be dependent on a static class structure — event properties shall be changed during runtime to provide the required flexibility. Secondly, events and event properties must couple in a transparent way, meaning that event properties and respective values must be read from and written to the database in such a way that it does not require a different SQL query for each type of event nor event-specific code. Rails’ ActiveRecord comes in handy but it doesn’t completely solve this issue, so further work will have to be done in order to store the event properties and respective values.

The Dynamic Object Model

The first attempt at solving the generalization issue starts by trying to apply the Dynamic Object Model design pattern [Riehle(2000)]. The intent of the Dynamic Object Model design pattern is to allow a system to have new and changing object types without having to reprogram the system. By representing the object types as objects, they can be changed at configuration time or at runtime, making it easy to change and adapt the system to new requirements. Figure 3.14 depicts the class diagram of the pattern.
In simple terms, the Dynamic Object Model decouples the concepts of objects (generally defined as `Component`), object types (`ComponentType`), properties (`Property`), property types (`PropertyType`) and property values (`Value`). This comes at the benefit of making it possible to dynamically (as in, in runtime) change the object structure without having to recompile or restart the application.

Applying the Dynamic Object Model to the events structure we would get the class diagram represented in Figure 3.15.
This model solves both requirements to generalize the application with enough flexibility. Data can be stored in the database in a very flexible way, discarding the need to have event properties stored as meta-information (i.e., as part of the table structure) but rather as information itself (by using the concept of Property and Value), and makes it easy to access the data back and forth by instanting Event objects with the properties and values.

A Rubyfied Dynamic Object Model

When the Dynamic Object Model was devised it was targeted (along with most of the GoF Design Patterns) at the non-interpreted and non-dynamic languages like Java and C++. Although a direct implementation of the Dynamic Object Model would solve our requirements to achieve a generalized and pluggable system, by using Ruby capabilities we can furthermore improve the flexibility and extensibility of the application.

First off, Ruby is dynamically typed and uses the concept of duck-typing. This means two things. First, attribute types are only checked at runtime and are not statically
defined so a variable can be (not at the same time, though) an Integer and a String afterwards. Secondly, duck-typing signifies that “if it squawks like a duck and walks like a duck, it’s a duck”. So, if a variable holds an Integer value, it’s an Integer.

The result of this is that there’s no need to have two separate classes PropertyType and Property. In the original Dynamic Object Model pattern, the PropertyType class was a façade to the class attributes so that getting and setting its values would not depend on its type. In Ruby we don’t need to cast variables, so a direct assignment is enough to change its type. Therefore, we can discard the PropertyType class and safely use the Property and Value class regardless of its type.

A Rubyfied and Really Dynamic Object Model

The author’s description of this design pattern is precise if we consider or assume that we only need to swap types but not create or destroy events at runtime. Considering this, changing types is a matter of setting the attribute that holds the ComponentType object in the Component instance and, reflexively, altering the Property List that holds properties and respective values.

If we intend to add or delete event types at runtime without interrupting the application we need to have a mechanism that allows such modifications to occur at runtime. One possible solution goes by storing the class attributes in a text file and then injecting them through the use of Ruby meta-programming capabilities. Instead of storing the event attributes in the class itself, these are stored in a formatted human-readable text file (YAML — YAML Ain’t Markup Language) that is parsed and interpreted at runtime. The property definitions are loaded into the respective EventType that is then used along with the Event object. The values of the properties are loaded from the database.

Ruby meta-programming capabilities further allow injection of code, meaning that we could create code to manipulate existent code, similarly to Aspect Oriented Programming but different in concept. But, again, this adds a lot of complexity to the application, having been therefore avoided.

Algorithms 1 and 2 show how a Sports event is represented in Ruby using a Ruby class and an YAML file to store the meta-information from its attributes, respectively.
Algorithm 1 The skeleton of an EventType class.

```ruby
class Soccer < Sports
  @properties
  def join_team(team, player)
  end
  def leave_team(team, player)
  end
  ...
end
```

Algorithm 2 The YAML file holding the properties for a Sports EventType class.

```
options:
- team1_players
- team2_players
attributes:
  team1_name:
    var_name: team1_name
    name: Name of Team1
type: text_field
    options:
      text_field_size: 30
```

Using a separate text file to store attributes (YAML) not only makes it possible to change it at runtime but also avoids cluttering the model with information that does not pertain to the model. As can be seen from Figure 2, information used in views like the type of HTML input field, its length and its name is stored there as well. Having to store such information directly in the model object would disrupt the concept of the Model-View-Controller pattern.

Mixing Ruby, Rails and the Dynamic Object Model

The Dynamic Object Model is an abstract pattern and therefore it’s desirable to have a general implementation that can be used without having to explicitly create its concrete class components. In Java or C++ this is not entirely possible — although abstract classes can define the skeleton and a common base to start off, they have to be separately inherited and explicitly created to give functionality to the objects.

In Ruby it is not only possible to dynamically create the class models but also advisable. This can be done by using Ruby mixins, a meta-programming mechanism that allows injecting modules (blocks of code that extend a class but are not class themselves nor can be instantiated into objects) into classes. Modules themselves can contain classes
Algorithm 3 Extending a Rails ActiveRecord model with the Dynamic Object Model pattern.

```ruby
class Event < ActiveRecord::Base
  include DynamicObjectModel
  def method
    ...
  end
end
```

that bind to the target class, meaning that a class that gets injected can immediately create dependent and nested classes.

To this purpose, we have conceived a Ruby library that can be mixed into any Rails ActiveRecord class. As previously explained in section 3.4.3, ActiveRecord maps the database tables into Ruby classes, providing methods to create and delete rows as well as to change attributes without having to explicitly create SQL queries. Our library reuses these concepts in order to group the Event, EventType and the Property List, meaning that a change in a property is immediately reflected in the Event object and in the persistency media (in this case, the database).

In order to make use of the Dynamic Object Model, one can do as follows in Algorithm 3.

By including the DynamicObjectModel mixin the ActiveRecord model will immediately be able to keep a list of properties that are inherited from the definition of the Event-Type and its respective YAML file. The values that these properties hold are also automatically managed and everytime an Event object is loaded from the database the property list will be fetched in a automated way. Updates to these values are also transparently saved, sparing the developer from digging through the class structure. `Object.property=` would be enough to update or set the value of a property.

Persisting the Data in the Database

Persisting the information in the database is a matter of mimmicking the object model diagram that’s represented in Figure 3.15 and improved according to what’s written in section 3.5.

Figure 3.16 depicts the actual database schema used to properly implement the design pattern.
3 Spateo - Conception and Development

**Instantiating the Dynamic Object Model**

There's just one step missing in order to make the system truly transparent: object instantiation. This problem is typically addressed by implementing a Factory Method [Erich Gamma(1994)] design pattern. The factory method design pattern handles this problem by defining a separate method for creating the objects, whose subclasses can then override to specify the derived type of product that will be created.

Modern languages that support reflection, like Ruby, Java, or C#, make it even easier to instantiate the objects according to their type. Ruby is capable of instantiating an object by using a `String` representation of the class name. For instance, in order to instantiate a new `Event` of `EventType` Soccer one could do `Kernel.const_get('Soccer').new`. Likewise, in Java one could do `Class.forName('Soccer').newInstance()`. This way the class name can be stored in the database along with the `Event` information and later on be used to instantiate the respective `EventType` class.

Whenever the object is instantiated, the YAML file containing all the attributes and respective meta-information is read and transparently incorporated into the `Event` object. From then on it's possible to access the attributes as instance variables. The ActiveRecord methods for loading and saving the data are overwritten in order to transparently save the object properties.

![Database Schema](image)
This object instantiation is fully mixed in with Rails ActiveRecord. The ActiveRecord::Base.find() method is extended in order to load the Event object from the database inside the correct EventType, along with its Properties list. Hence, Event.find(id) will load the event information into the correct EventType class and make all its properties (that are stored, as we’ve seen, in separate tables) available through simple methods like event.property() or event.property=. Furthermore, all the create, save, update and destroy actions will update the event properties in a transparent way without requiring the developer to do it explicitely. An event.save() call will not only save the information related to the Event object but all the Properties that might have been changed.

Wrapping Up

In order to generalize the system so it can support multiple event types and properties without requiring extra code, we’ve started by implementing the Dynamic Object Model, a design pattern that separates the concepts of objects and objects types so that they can be changed during runtime.

By making use of the meta-programming and reflexion capabilities of Ruby we’ve cut the complexity of the pattern and its instantiation methods, resulting in a system that can automatically instantiate any kind of Event without having to have specific code to support a new EventType.

Furthermore, we’ve abstracted the properties and its meta-information out of the classes so that it can be changed effortlessly during runtime. Such information is stored in YAML text and human readable files.

Finally, we’ve combined the design pattern into a library that blends with the Rails’ ActiveRecord framework. This way, all the event data can be stored and loaded from the database seamlessly, without requiring one single line of SQL, as if it had always been an object.

This approach has made it possible to keep a very flexible system that can be manipulated at runtime and instantaneously reflect changes onto the events. Furthermore, no specific code is added to the application or to the database, meaning that the deletion or creation of new event types does not require any code refactoring or any separate tables to support them.

The description of Algorithm 4 gives a brief overview of a sample usage:
Algorithm 4 Sample usage of the Dynamic Object Model design pattern in Spateo.

```ruby
# instantiate a new event of type Soccer
soccer = Soccer.new
# the object hasn't any property set...
# (this method is inherited from the Sport super class)
puts soccer.team1_players # => []
# but the meta-information is available
# this is used for the views
puts soccer.options['team1_color']['field_type'] # => 'text_field'
# (this method is inherited from the Event super super class)
puts soccer.attendees # => []
# we can add a player to the event
soccer.add_attendee('Mario', :rsvp => 'attending')
# or set a property that's related to the event type
soccer.max_num_players = 22
# let's save all the information into the database
soccer.save # id => 1
# loading it will automatically restore all the information
soccer = Soccer.find(1)
puts soccer.max_num_players # => 22
puts soccer.attendees # => ['Mario']
```
4 Prototype

In this chapter we’ll guide the reader through the features that have been previously defined. We’ll cover the common features found on most social platforms and those typical of an events applications. Furthermore, we also show how our approach to events customization looks like. This is the core functionality that drives our proposal apart from the other competitors.
In Figure 4.1 we can see the website’s entrance page. Here we decided to give an immediate answer to what’s the website about — a place where you can plan your events. From here on, users can signup or login. Notwithstanding, and considering that users are getting overwhelmed with registrations, we don’t force or impose the creation of an accounts. Users can navigate freely and even RSVP to an event without an explicit registration. Further functionality is only disclosed to registered users (like the ability of connecting with friends) but we pose no obligation to being registered. It’s up to the users to decide whether they want to be registered or not.

From a non-functional perspective, we’ve decided to keep a frontpage that’s not flooded with advertising and irrelevant information. We sum up in a sentence what’s the website
all about and we entice users into continue reading by showing pictures with vivid colors along with a brief description of what they’ll get.
4 Prototype

4.2 Session management

4.2.1 Signup

Figure 4.2: Create an user account.

Figure 4.2 depicts how’s the registration process. The user is asked to provide a few personal details, choose a password and specify his location. Further on, after the user registration process, the user will be able to provide more information about himself. Provider registrations have one further step where they can specify the kind of service they provide and/or details about their venue. Lastly, the captcha (“Completely Automated Public Turing test to tell Computers and Humans Apart”) is used to avoid automated spamming registrations. This technique is effective but it holds some problems with people that carry visual disorders. We’re considering replacing this method later on.
4 Prototype

4.2.2 Login

Figure 4.3: Log in on Spateo.

The login process that’s depicted in Figure 4.3 is very common in the Internet. One provides a unique identification token (like a username or email) and the respective password.

We’ve decided to make this a separate action (and hence the modal dialog that prevents any further action) because the login is an atomic action that does not interact with any other part of the page. In case of a successful login, the modal dialog box disappears and the user can continue using the website. If the login fails, a message is presented to the user indicating it.

58
4.3 Social features

4.3.1 Profile

Logged in users have their own profile page, as seen in Figure 4.4. The existence of a profile page is a fundamental part of any social website. Here, we allow users to disclose a little bit more about themselves, expose the groups they belong to, their public photo albums, profile messages and their upcoming event. They can also share their status (what they’re currently doing) with other people.
4.3.2 Albums/Photos

Registered users are able to create their photo albums and upload photos through the interface in Figure 4.5. This is especially useful for sharing pictures of a past event with the event attendees.
4.3.3 Friends

The friends page that can been seen in Figure 4.6 shows users that have been marked as friends or a certain user. This permits one to see the network of friends and perhaps find bridges through common friends.

Figure 4.6: Friends listing.
4.3.4 Inbox

Figure 4.7: The message center where users can read incoming messages, compose new messages and check for invitations.

In the message center (Figure 4.7) users are able to check their inboxes for incoming messages, consult the messages they have sent, check for event invitations or friend requests. Also, users are able to compose new messages.
4.3.5 Compose a message

As part of the messaging exchange mechanism, users can use a very simple interface to compose their messages and send to their friends. Figure 4.8 depicts the interface used to compose messages.

Figure 4.8: “Compose a new message” form.
4.3.6 Groups

Another common feature of social networks is that of groups. Groups, as the name suggests, are a gathering of people that share similar interests. In the particular case of our platform, groups are also a handy feature for sharing events with several people that might be interested.

Figure 4.9: A page of a group.
4 Prototype

4.4 Events Platform

4.4.1 Create an event

The event creation process is composed of five simple steps.

The first step, seen in Figure 4.10, is used to define the kind of event. This is of utmost importance as it will set how the event can be customized in subsequent steps.

![Create an event - step 1 of 5.](image-url)
Step 2 of the event creation process asks for information transversal to all kind of events. Generally, the title of the event, date, a flyer or poster picture and whether the event is private (not searchable) or whether the user wants to restrict access to the event page by the means of setting a password. Picture 4.11 shows this phase of the event creation process.

Figure 4.11: Create an event - step 2 of 5.
Step 3 is where our proposal sets apart from the other offerings currently available. More than allowing to set the address of a venue, we provide a system capable of searching for (registered) venues, in case the user hasn’t decided on one yet. In the particular case of Figure 4.12, the user is planning a dinner and, hence, looking for a restaurant. We ask him a few questions about the location of the restaurant and, furthermore, some requirements he might want to fulfil, namely if it has vegetarian meals or not or if there’s a smoking reserved area or not. Optionally, the user can skip this step altogether if he already knows of a venue (tab “I already have a venue for this event”) or in case the venue is not registered on the system he can simply put in the address.
Step 4 is for personalizing the event according to the kind of event one has chosen. As represented in Figure 4.13, the user is able to add in information that might be useful to potential attendees, like the estimated cost or the menu available.
The last step of the event creation process, as depicted in Figure 4.14, consists of previewing how the event page will look like. Errors can still be corrected by going back. In case the user is satisfied with the result, he can simply finish it and automatically publish it.
4 Prototype

4.4.1.1 Manage the event

Figure 4.15: Manage an event.

After the event creation process, the event is set live and accessible by anyone (provided that the user hasn’t set it to private or defined a password). Figure 4.15 shows how an event page looks like and the functionality that’s available to the event creator/manager. As can be seen, the user is able to edit the event details (those previously specified in steps from 1 to 5), change the theme, contact attendees for critical updates, export the attendee list, invite friends or post the event to his groups.
4 Prototype

4.4.1.2 Event page

Some events provide further functionalities to its attendees. Events of the type ‘Soccer’ like the one shown in Figure 4.16 display a soccer field where users can visually define teams upon RSVPing to them. This degree of customization is exclusive to our proposal.
One of the flag features of any event application is that of RSVPing. Tracking which users are planning on attending the event is of crucial importance to the event organizers. We made this task as effortless as possible. We do not demand any registration in order to say whether the user is attending the event or not. Users are invited to finish their registration process later on. Figure 4.17 exemplifies how one can RSVP to an event.
4 Prototype

4.4.2 Search

Another flagship feature of such an application is the ability of searching for a particular thing. In our application we provide the tools for searching for users, groups and looking up events. Figure 4.18 depicts how one can search for other users by their name or email and filter by their location.

![Figure 4.18: Search for other people.](image)
4.4.3 Themes

Theming is not a crucial feature but it’s definitely an add on to the whole value of an application. Differentiating events by the means of a visual appearance might sound irrelevant at first, but it adds a personal touch to it and users are certainly candid of such feature. Figure 4.19 shows some themes that are available to be used on the events pages. Themes can also be used on profile pages.

Figure 4.19: Pick a theme for the profile page or for an event.
4 Prototype

4.4.3.1 Theme creation

We could have taken the simpler approach of making available only a strict set of themes but we think that letting users create their own themes is of enormous value to them. The interface shown in Figure 4.20 is used to compose themes for the events. MySpace allows users to insert random HTML and CSS, which makes it more powerful but create a horde of not valid and inaccessible web pages. By following this method we only allow users to change visual aspects of the page without violating any W3C standards or deteriorating accessibility.

Figure 4.20: Create a new theme.
5 Conclusions and Further Work

In order to draw proper conclusions one must build upon the premises subject to his proposal. What we’ve hereby proposed was to technically conceive an events planning web application on top of a social platform. Furthermore, we also devised non-functional requirements that pertain to successful applications but are not immediately noticeable. Conceiving a simple and usable interface, guaranteeing good performance levels or maintaining a layer of security to ensure the user’s privacy were some of those goals. Also, making sure the features we have devised work as expected and, hence, a test policy.

As far as the aforementioned goals that were set for the alpha stage concerns, these have been achieved. All the functional and non-functional requirements as well as the testing policies have been successfully implemented. We’ve reached a fully-working system capable of executing its core functionality: help to plan, manage and organize events and provide a social network whereto users can wield and use to their benefit of advertising their events.

From an engineering and pure technical level, we believe we have devised and elaborated a proposal that goes in accordance with the objectives defined and adds a good value to the market. From a business perspective, something that we’ve valued and consider crucial to the full success of such project, it’s not reasonable yet to draw any conclusions up to this date. Due to the complexity of the project and the lack of more human resources, it has been undoable to launch the project in time for assessing whether it’s a commercial success or not. The author is in full knowledge of this and will therefore postpone drawing any conclusions as to whether the unequivocal favorable outcome of the project was attained.

As for further work and areas of improvement, we realize that such a project as the one hereby proposed is far from being ever completed and reaching a stage of perfect stability. On this thesis we’ve covered all the planning and development phases up to the alpha version. This intermediate version is typically used for deploying all the expected functionalities and conducting internal tests within a controlled environment. This stage is provided without any guarantee whatsoever as to what data loss concerns.

Up to this stage, we’ve been concerned with implementing the list of functional and non-functional requirements that have been previously established. Following stages will be focused on bug tracking and bug correction and assessing and validating the list of functional and non-functional requirements. Functional requirements can be evaluated by
observing the user’s interest in a specific feature. Non-functional requirements demand a more complex analysis of the data retrieved by conducting usability tests and surveys. Evaluating whether an user interface is suitable to our target demographics requires continuous observation of how users actually put the application to use.

As the application leaves the alpha stage to commence the beta stage, an increase in the user base is expected during the course of opening it up to the general public. This growth will certainly demand more performance in order to answer to the increase in the number of requests per second. Two non-exclusive strategies can be taken: inject more hardware or optimize the software. Since hardware is costly and the budget is already considerably squeezed, priority goes to performance optimization that requires nothing more than expertise and time. There are a lot of ways of optimizing such an application. Often, premature optimization is more harmful than advantageous, as effort will be put in blocks of code that, unless plain badly coded, might not be the application biggest bottlenecks. Therefore, performance optimization and sequential scalability is done by observing the bottlenecks and fine-tuning them. Typically, 3-thier client-server applications die on the database or on the application server. Strategies for overcoming these issues usually go by adding cache servers (like memcached or Squid) to avoid requests reaching the application server and the database and/or building a cluster of databases to deal with a surge in read/write requests.

Whenever the application reaches a stable phase where no major hiccups are found and bug reports diminish visibly, it’s time to think about new features. On the top of the list is the creation of a mobile/iPhone version. Having the possibility of checking for events nearby ones current location — perhaps resorting to the iPhone’s internal GPS device to track the coordinates — might be of immense potential to party goers and other event attendees. Considering how the application was developed, it’s easy to reuse the application programming interface and simply conceive an interface that’s suitable for mobile devices. Furthermore, the application programming interface (API) might also be exposed to general use, although no plans have been considered for up to this date.

Opportunities to improve the application are unlimited. Our strategy is to observe with close proximity how the users use the application, what they expect out of it and how competitors react to a new player on the market. Offering the best solution is only the beginning. Striving also depends on offering what users want to use.
Bibliography


78
A Analysis and Specification of Functional and Non-Functional Requirements, and Test Policies of the “Spateo” Application
Functional Requirements

User and Profile Management
The system shall provide a user registration form, a user login and logout form and an access list to filter which users can access which pages. For instance, events ought to be created only by duly registered and authenticated users. Failure to do so must carry with an invitation to the user to proceed with the registration on the system.

Furthermore, the user must be able to customize its profile by inserting information like pictures, interests, etc..

<table>
<thead>
<tr>
<th>Requirement #: F/UM/1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Depends on</td>
<td>-</td>
</tr>
<tr>
<td>Priority</td>
<td>1</td>
</tr>
<tr>
<td>Title</td>
<td>User registration</td>
</tr>
<tr>
<td>Description</td>
<td>A form shall be provided to the user so that he can proceed with registration on the website. The following items must be recorded:</td>
</tr>
<tr>
<td></td>
<td>• First and Last name</td>
</tr>
<tr>
<td></td>
<td>• Email</td>
</tr>
<tr>
<td></td>
<td>• Password</td>
</tr>
<tr>
<td></td>
<td>• Country</td>
</tr>
<tr>
<td></td>
<td>• City</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/UM/2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Depends on</td>
<td>F/UM/1</td>
</tr>
<tr>
<td>Priority</td>
<td>2</td>
</tr>
<tr>
<td>Title</td>
<td>Confirmation of email upon user registration</td>
</tr>
<tr>
<td>Description</td>
<td>The user email must be confirmed before properly activating the user account. This shall be done by sending an email to the user with the activation link that the user must thereafter click.</td>
</tr>
<tr>
<td>Requirement #: F/UM/3</td>
<td>Depends on</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------</td>
</tr>
<tr>
<td>Priority</td>
<td>1</td>
</tr>
<tr>
<td>Title</td>
<td>User login</td>
</tr>
<tr>
<td>Description</td>
<td>Upon user registration and activation of the account, the user must be able to login on the website by typing its email and password.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/UM/3</th>
<th>Depends on</th>
<th>F/UM/1, F/UM/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Password recovery</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>In case the user has forgotten its password, a mechanism of resetting its password must be available. This should work by sending a reset password link to the email and proceeding from within the website thereafter.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/UM/3-1</th>
<th>Depends on</th>
<th>F/UM/1, F/UM/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Password recovery - Reset link</td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Upon clicking on the link sent to the user email address, a new form shall be prompted so that he can introduce a new password.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/UM/4</th>
<th>Depends on</th>
<th>F/UM/1, F/UM/2, F/UM/3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>User profile</td>
<td></td>
</tr>
</tbody>
</table>
### Requirement #: F/UM/4

**Description**

Upon signing up, confirming the email and signing in the user will be shown his user profile where he can change his details.

The user will then have access to the following functionalities:

- Settings its Status
- Personal Details
- Featured Event (if any)
- Calendar
- Friends
- Groups
- Profile Comments
- Photo Albums

### Requirement #: F/UM/4-1

<table>
<thead>
<tr>
<th>Depends on</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Priority</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>Title</strong></td>
<td>User profile - Setting the status</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>It shall be possible for the user to set its status in a Twitter-like way.</td>
</tr>
</tbody>
</table>

### Requirement #: F/UM/5

<table>
<thead>
<tr>
<th>Depends on</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Priority</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Title</strong></td>
<td>Privacy settings</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The user will be able to set whether his profile is public or private, if he/she wants to receive email notifications from anyone or only from real friends and whether he/she wants to receive a monthly newsletter.</td>
</tr>
</tbody>
</table>

### Requirement #: F/UM/6

<table>
<thead>
<tr>
<th>Depends on</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Priority</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>Title</strong></td>
<td>Themes</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The users shall be able to create themes and customize them or select one from a list of already existent themes to use in their profile.</td>
</tr>
<tr>
<td>Requirement #: F/UM/6-1</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Depends on</strong></td>
<td>F/UM/6</td>
</tr>
<tr>
<td><strong>Priority</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>Title</strong></td>
<td>Themes - Publishing themes</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The user will be able to submit his theme to be approved and, if so, published for public use.</td>
</tr>
</tbody>
</table>
Events Management

The events management functional module can be divided into several sub-modules. This module is the core of the application and should provide users with a system capable of adding, editing, removing and managing events. Moreover, the system must be capable of providing specific functionality depending on the type of event that's being inserted. For instance, if the user is planning a dinner he must be able to specify not only the location and venue but also what kind of food will be available, estimated cost for the dinner, etc..

Further functionalities shall also be available, like customized search of venues. The event organizer shall be able to search for venues that comply with his preferences, like searching for restaurants that have vegetarian meals on their menus.

**Requirement #: F/EM/1-P**

<table>
<thead>
<tr>
<th>Depends on</th>
<th>F/UM/3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>1</td>
</tr>
<tr>
<td>Title</td>
<td>Insertion of a new event</td>
</tr>
<tr>
<td>Description</td>
<td>Pseudo functional requirement placeholder for handling sub-requirements.</td>
</tr>
</tbody>
</table>

**Requirement #: F/EM/2**

<table>
<thead>
<tr>
<th>Depends on</th>
<th>F/EM/1-P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>1</td>
</tr>
<tr>
<td>Title</td>
<td>Definition of the Event Category and Sub Category</td>
</tr>
</tbody>
</table>
| Description  | When proceeding with the event registration, the system shall ask for the category (ie, Party, Social Meeting, etc..) and the sub category of the event (ie, Birthday Party, Conference, etc..).

The information given will determine how the customization of the event insertion is done.

**Requirement #: F/EM/3**

<table>
<thead>
<tr>
<th>Depends on</th>
<th>F/EM/1-P, F/EM/1-P, F/EM/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>1</td>
</tr>
<tr>
<td>Title</td>
<td>Insertion of the basic details about the event</td>
</tr>
</tbody>
</table>
| Description  | Upon determination of the category and sub category (see F/EM/2), the application shall ask for basic information about the event, namely:

- Event start and end date
- Event title
- Event venue

Furthermore, it shall ask whether the event needs a password to be accessed or/and if it's private (ie, it's not searchable).
<table>
<thead>
<tr>
<th>Requirement #: F/EM/4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depends on</td>
</tr>
<tr>
<td>Priority</td>
</tr>
<tr>
<td>Title</td>
</tr>
<tr>
<td>Description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/EM/5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depends on</td>
</tr>
<tr>
<td>Priority</td>
</tr>
<tr>
<td>Title</td>
</tr>
<tr>
<td>Description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/EM/5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depends on</td>
</tr>
<tr>
<td>Priority</td>
</tr>
<tr>
<td>Title</td>
</tr>
<tr>
<td>Description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/EM/6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depends on</td>
</tr>
<tr>
<td>Priority</td>
</tr>
<tr>
<td>Title</td>
</tr>
<tr>
<td>Description</td>
</tr>
</tbody>
</table>
### Requirement #: F/EM/7

**Depends on**
F/UM/1-P

**Priority**
1

**Title**
Visualization of a previously created event

**Description**
The event must be viewable by the user, regardless of whether he's authenticated or not.

### Requirement #: F/EM/8

**Depends on**
F/UM/7

**Priority**
1

**Title**
Request password for protected events

**Description**
When the events require password a modal dialog shall be presented to the user so he can introduce the password. In case of success, the application shall open the event show page (see F/UM/7).

### Requirement #: F/EM/9

**Depends on**
F/UM/1, F/EM/7

**Priority**
1

**Title**
Event templating

**Description**
The user must be able to further customize its event page by selecting from a base of already created design styles or create its own.

### Requirement #: F/EM/9-1

**Depends on**
F/EM/9

**Priority**
1

**Title**
Select an event template from a list of given templates

**Description**
The user shall be able to select a template from a list of previously created design templates and also preview its usage.

### Requirement #: F/EM/9-2

**Depends on**
F/EM/9

**Priority**
1

**Title**
Create a new event template design

**Description**
The user shall be able to create his own template design by defining key CSS styles and key images. Constraints shall apply.
<table>
<thead>
<tr>
<th>Requirement #: F/EM/10</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Depends on</td>
<td>-</td>
</tr>
<tr>
<td>Priority</td>
<td>1</td>
</tr>
<tr>
<td>Title</td>
<td>Contact attendees</td>
</tr>
<tr>
<td>Description</td>
<td>The user shall be able to contact through email the attendees.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/EM/11</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Depends on</td>
<td>-</td>
</tr>
<tr>
<td>Priority</td>
<td>1</td>
</tr>
<tr>
<td>Title</td>
<td>Export attendees list</td>
</tr>
<tr>
<td>Description</td>
<td>The user shall be able to export the attendees list (along with emails) to a CSV (comma separated values) file.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/EM/12</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Depends on</td>
<td>-</td>
</tr>
<tr>
<td>Priority</td>
<td>1</td>
</tr>
<tr>
<td>Title</td>
<td>Invite friends to event</td>
</tr>
<tr>
<td>Description</td>
<td>The user shall be able to contact friends (registered or not) for the event. <strong>NOTE</strong>: This action shall trigger an email to the invitees.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/EM/13</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Depends on</td>
<td>-</td>
</tr>
<tr>
<td>Priority</td>
<td>1</td>
</tr>
<tr>
<td>Title</td>
<td>Post event to groups</td>
</tr>
<tr>
<td>Description</td>
<td>The user shall be able able to post the event to groups he’s part of.</td>
</tr>
</tbody>
</table>
# Events Browsing

The system must provide with a way of consulting events depending on location, date and venue. Furthermore, the physical location of a user (country, city) shall be automatically determined or manually retrieved within the process of registration. Based on this information, the system shall present a listing of events happening on that area or on nearby areas in case there’s none on that particular area.

<table>
<thead>
<tr>
<th>Requirement #: F/EB/1</th>
<th>Depends on</th>
<th>Priority</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Browsing events</td>
<td>A registered user shall be able to browse its past and upcoming events as well as events happening in a specified location.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/EB/1-1</th>
<th>Depends on</th>
<th>Priority</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Listing upcoming events</td>
<td>The first page of the website should display upcoming events taking place near the user specified location or, optionally, by selecting a certain location.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/EB/2</th>
<th>Depends on</th>
<th>Priority</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Search for events</td>
<td>A tool will be provided in order to search for events by its title and/or description and also filter them by city and/or country.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/EB/3</th>
<th>Depends on</th>
<th>Priority</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Friends’ events</td>
<td>The system should provide with a simple way of consulting the events that friends will be attending.</td>
</tr>
</tbody>
</table>
### Requirement #: F/EB/4

<table>
<thead>
<tr>
<th>Depends on</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>1</td>
</tr>
<tr>
<td>Title</td>
<td>Nearby Events</td>
</tr>
<tr>
<td>Description</td>
<td>The system shall provide a listing of events happening nearby the user, by the means of his geolocation that can set during the registration process or used as a filter for listing the events.</td>
</tr>
</tbody>
</table>

### Requirement #: F/EB/5

<table>
<thead>
<tr>
<th>Depends on</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>1</td>
</tr>
<tr>
<td>Title</td>
<td>Popular Events</td>
</tr>
<tr>
<td>Description</td>
<td>The system shall provide a listing of events that are popular, either by getting a lot of page views or due to high attendance rate.</td>
</tr>
</tbody>
</table>
**Social Network**

The application should provide the typical functionality of a social network, namely being able to create circle of friends, send messages within users, create common interest groups, share pictures, etc. Furthermore, the social network shall be used as the means to further extend the events functionality by allowing users to advertise their events among friends, post them to groups, etc..

<table>
<thead>
<tr>
<th>Requirement #: F/SN/1</th>
<th>Depends on</th>
<th>Priority</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F/UM/3</td>
<td>1</td>
<td>Adding a user as friend</td>
<td>Users can be added as friends. They shall receive a notification and act upon by accepting or rejecting. If they accept, the user shall be listed in the group of friends.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/SN/2</th>
<th>Depends on</th>
<th>Priority</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F/UM/3</td>
<td>1</td>
<td>Sending a message to a user</td>
<td>A user can send a message to another user.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/SN/2-1</th>
<th>Depends on</th>
<th>Priority</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>1</td>
<td>Sending a message to several users</td>
<td>A user can send a message to multiple recipients.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/SN/3</th>
<th>Depends on</th>
<th>Priority</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F/UM/3</td>
<td>1</td>
<td>Checking the list of messages (INBOX)</td>
<td>Each user has an INBOX for incoming messages and SENT folder with all the messages that were sent to other users.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: F/SN/3-1</th>
<th>Depends on</th>
<th>Priority</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>1</td>
<td>Deleting messages</td>
<td>A user can delete previously received messages.</td>
</tr>
<tr>
<td>Requirement #: F/SN/4</td>
<td>Depends on</td>
<td>Priority</td>
<td>Title</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------</td>
<td>----------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>Group management (dummy)</td>
<td>This is a placeholder for requirements related to groups.</td>
</tr>
<tr>
<td>Requirement #: F/SN/4-1</td>
<td>F/UM/3</td>
<td>1</td>
<td>Create a group</td>
<td>A user can create a group.</td>
</tr>
<tr>
<td>Requirement #: F/SN/4-2</td>
<td>-</td>
<td>1</td>
<td>Join a group</td>
<td>A user can join a previously created group.</td>
</tr>
<tr>
<td>Requirement #: F/SN/4-3</td>
<td>-</td>
<td>1</td>
<td>Leave a group</td>
<td>A user can leave a previously joined group.</td>
</tr>
<tr>
<td>Requirement #: F/SN/4-4</td>
<td>-</td>
<td>1</td>
<td>Group forum</td>
<td>A minimalist forum shall be available for users to talk to each other on the group.</td>
</tr>
<tr>
<td>Requirement #: F/SN/5</td>
<td>-</td>
<td>1</td>
<td>Manage invitations</td>
<td>A user shall be provided with an interface to read, delete and respond to his invitations.</td>
</tr>
</tbody>
</table>
### Requirement #: F/SN/6

**Depends on**  
-

**Priority**  
1

**Title**  
Photo Albums management (dummy)

**Description**  
This is a placeholder for requirements related to photo albums.

#### Requirement #: F/SN/6-1

**Depends on**  
F/UM/3

**Priority**  
1

**Title**  
Create an album

**Description**  
A user can create albums to deposit photo pictures.

#### Requirement #: F/SN/6-2

**Depends on**  
-

**Priority**  
1

**Title**  
Upload photo pictures

**Description**  
A user can upload photo pictures to a album that belongs to him.

#### Requirement #: F/SN/6-3

**Depends on**  
F/SN/6-2

**Priority**  
1

**Title**  
Move photos between albums

**Description**  
A user can move photos between albums.

#### Requirement #: F/SN/6-4

**Depends on**  
-

**Priority**  
1

**Title**  
Delete an album

**Description**  
A user can delete an album along with all its photos.
## Usability and User Experience

<table>
<thead>
<tr>
<th>Requirement #: NF-1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Priority</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Title</strong></td>
<td>Complimentary actions shall be provided within a modal dialog</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Actions that do not require a significant change in the User Interface do not require a whole web page to load and can be duly substituted by modal dialogs. For instance, logging in is an action that is complimentary and only requires a small form. Therefore, instead of having a whole page just for doing the login and, furthermore, having to disrupt the user from its workflow activity, the system shall only present him with a modal dialog to proceed with the authentication.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement #: NF-2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Priority</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Title</strong></td>
<td>All user actions must return a system feedback</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>All actions pertained by the user must have a feedback by the system to show responsiveness. For instance, if the user improperly set a date, an error message shall be presented to the user. Another example: if the user finishes inserting a new event, an information message shall also be presented to the user acknowledging the insertion of a new event.</td>
</tr>
</tbody>
</table>

## Web Standards

<table>
<thead>
<tr>
<th>Requirement #: NF-3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Priority</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>Title</strong></td>
<td>The website must conform to CSS2 and XHTML 1.0</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The final output of the application must conform to CSS2 and XHTML 1.0. If necessary, non CSS2 and XHTML 1.0 compliant hacks may be used to make the website work in IE6 but only if strictly necessary.</td>
</tr>
</tbody>
</table>
Performance and Optimization

Performance can be seen as a balance between speed and scalability. Sometimes faster operations can consume more memory, which clearly deteriorates scalability. Therefore, performance-wise improvements must be pondered in terms of speed and scalability.

Also, Ruby on Rails is a very complete and powerful web framework but it's still not performance-oriented. Ruby itself also relies on an interpreter lacking performance-wise optimizations. Therefore, special caution must be given to heavy computational calculations and coding assumptions.

In this section we provide non functional requirements in order to achieve a better overall application. Measurements will be made and bottlenecks optimized further on, but these requirements are a good starting point.

### Requirement #: NF-4

**Priority**: 1

**Title**: Eager Loading

**Description**: Dynamic web pages typically require a lot of database queries to retrieve information from different tables. The sum of all these single queries is costlier than having an eager query retrieving all the required tables and columns at once.

In Rails, this can be easily achieved by doing:

```
Article.find(:all, :include => :author)
```

### Requirement #: NF-4.1

**Priority**: 2

**Title**: Piggy Backing

**Description**: Requirement #: NF-4 can be further improved by using a technique called piggy backing. This can be easily done by hardcoding the SQL JOIN code on the query itself, i.e.:

```
Article.find(:all, :conditions => "a.user_id=u.id", :joins => "a, users u")
```

### Requirement #: NF-5

**Priority**: 1

**Title**: Use SQLSessionStore or MemCached Session Management

**Description**: Typically, Rails uses Pstore to store the session data. It's typically very slow so the use of database session management is recommended (along with a very fast engine like MyISAM). Optionally, MemCache session management is also very fast but maintenance operations can be cumbersome.
Requirement #: NF-6

Title: Avoid the usage of costly on-the-run operations

Description: Rails allows the usage of handy helper methods to access the database models. For instance, finding by a column name can be written as `Model.find_by_column`. But generating the method handler on the go is a very expensive operation. Therefore, the usage of `Model.find(:conditions => ["column = ?", column_val])` is recommended.

Optionally, one can manually implement the method `find_by_column` on the model as a short-circuit.

Requirement #: NF-7

Title: Page caching

Description: Static information should always be cached. Dynamic information shall be governed by an expiration date specifying when it's outdated and requires an update.

For instance, one might use

```ruby
@expensive_calculation ||= begin
  perform_expensive_calculation_here
end
```

This way `@expensive_calculation` can get cached and another expensive calculation is therefore avoided.

Requirement #: NF-8

Title: Avoid heavy helper methods

Description: Some helper methods are provided to keep the HTML code cleaner or to provide some extra flexibility. Some of these methods are slow, so hardcoded HTML should be used instead.

For instance, instead of `start_form_tag :url => "/c/action"` one might use `<form action="/c/action">` instead.
<table>
<thead>
<tr>
<th>Requirement #: NF-9</th>
<th>Priority</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Rails provide some amenities to the programmer that be costly. For instance, instead of render:controller =&gt; &quot;greeter&quot;, :action =&gt; &quot;hello_world&quot; one should use render:controller =&gt; GreeterController, :action =&gt; &quot;hello_world&quot; which avoids calling the inflector to figure out the controller class name. Another example is the use of the pluralize() helper method. Instead of doing pluralize(n, 'post') you can provide the plural with pluralize(n, 'post', 'posts') and avoid a new inflector.</td>
</tr>
<tr>
<td>Requirement #: NF-10</td>
<td>Priority</td>
<td>Title</td>
<td>Use column indexes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Whenever table columns are used as filter criteria on an SQL WHERE clause, they should be defined are indexes if they’re not primary keys.</td>
</tr>
<tr>
<td>Requirement #: NF-11</td>
<td>Priority</td>
<td>Title</td>
<td>Use views whenever possible</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Instead of passing a very complex SQL query, try using views whenever possible. These are stored on the database and are faster.</td>
</tr>
<tr>
<td>Requirement #: NF-12</td>
<td>Priority</td>
<td>Title</td>
<td>Compress the output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This can be done by adding gzip compression as an after_filter on the Rails application,</td>
</tr>
</tbody>
</table>
Internationalization and Localization

Requirement #: I18N-1
Priority 1
Title Multi-language support
Description The application must support several languages by including dictionary files that can be used to translate the text.

Security and User Privacy

Requirement #: SUP-1
Priority 1
Title XSS
Description A whitelist mechanism will be devised in order to prevent Cross Site Scripting.

Requirement #: SUP-2
Priority 1
Title Privacy invasion protection
Description A layer will be built on top of user input to prevent users from exploiting and using application information in order to inject or modify information into other's users account.
User and Profile Management

Test #: T/UM/1
Refers to F/UM/1
Title User registration
Description Unit and functional tests should be done to cover the following cases:

1. Incomplete form. Expected result: fail;
2. Complete form and invalid entries. Expected result: fail;

Test #: T/UM/2
Refers to F/UM/3
Title User login
Description Unit and functional tests should be done to cover the following cases:

1. Invalid username/password. Expected result: fail;
2. Empty fields. Expected result: fail;
3. Correct username and password. Expected result: success

Test #: T/UM/3
Refers to F/UM/3
Title Password recovery
Description Unit and functional tests should be done to cover the following cases:

1. Empty fields. Expected result: fail;
2. Email is registered as pertaining to a user. Expected result: success and email sent to the given email address
<table>
<thead>
<tr>
<th>Test #: T/EM/4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Refers to</td>
<td>F/EM/3</td>
</tr>
<tr>
<td>Title</td>
<td>Insertion of the basic details about the event</td>
</tr>
</tbody>
</table>
| Description   | Unit and functional tests should be done to cover the following cases:  
1. Incomplete form. Expected result: fail;  
2. Complete form and invalid entries. Expected result: fail;  

<table>
<thead>
<tr>
<th>Test #: T/EM/5</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Refers to</td>
<td>F/EM/6</td>
</tr>
<tr>
<td>Title</td>
<td>Elimination of a previously created event</td>
</tr>
</tbody>
</table>
| Description   | Upon elimination the record should be cleared from the database. Test case:  
1. Database table does not contain the record. Expected result: success. |
### Events Infrastructure

**Test #: T/EI/1**

<table>
<thead>
<tr>
<th>Refers to</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Title</strong></td>
<td>Event attributes infrastructure</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Different events have different attributes. Tests will be needed to ensure that the solution engineered to address this works correctly. This means creating events, setting attributes, getting attributes and making sure they’re correctly written on the database and all the meta-information is preserved.</td>
</tr>
</tbody>
</table>

**Test #: T/EI/1-1**

<table>
<thead>
<tr>
<th>Refers to</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Title</strong></td>
<td>Event attributes infrastructure II</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Event attributes are fetched from YAML files containing meta information about the event. A unit and behavioral test shall be provided to ensure that attributes are correctly read.</td>
</tr>
</tbody>
</table>

**Test #: T/EI/2**

<table>
<thead>
<tr>
<th>Refers to</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Title</strong></td>
<td>Event inheritance</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Events can inherit between each other (for instance, Soccer inherits from the generic event Sports). Attributes, properties and method overrides will be inherited as well. Unit tests shall be created in order to ensure that this process happens as expected.</td>
</tr>
</tbody>
</table>