



# Grays 2.0: A Web 2.0 on-line Medical Social Bookmarking system using the Google App Engine and Django

A Dissertation

Submitted to

The Division of School of Computing

Dublin Institute of Technology

In Partial Fulfillment

of the Requirements for the Award

MSc in Computing (Information Technology)

Juan D. Convers

May 2009

# Acknowledgments

First and particularly, I would like to thank the big support and encouragement I received from my supervisor, Damian Gordon of the School of Computing at the DIT during the development of this project. He not only helped me to define a good structure for this project, provided me very good advice in every session to improve the dissertation and was flexible with my tight schedule during this period of the year. I only have words of gratefulness for him.

I would like to thank my employer, Google Ireland and in particular my department Search Quality Evaluation for being flexible with the time I needed to finish up this dissertation project. The study leave they granted during this period was fundamental to finish this project on time.

I also would like to thank all the users of the medical social bookmarking application who filled out the feedback form. Their help was invaluable to write the Results chapter.

Finally I would also like to thank all my family members and friends who encouraged me and supported me during this thesis period.

# Declaration

I certify that this dissertation which I now submit for assessment for the award of MSc in Computing (Information Technology), Dublin Institute of Technology, is entirely my own work and has not been taken from the work of others, the dissertation was prepared according to the regulations for postgraduate study of the Dublin Institute of Technology and the dissertation has not been submitted for evaluation for any academic purpose other than in particular fulfilment for the stated above.

Signed

---

Juan D. Convers

Date: May 6 / 2009

# Abstract

Even if at beginning the term Web 2.0 ignited a lot controversy and it was considered by some as a market ploy, during the last few years the Web 2.0 has demonstrated it is a reality and web services like blogs, wikis, podcasts, social networks and social bookmarking systems are used everywhere. The Ajax set of technologies are common in many services and certain features that once were a novelty like the Tag Clouds, the user-generated contents and ranking and the online social interaction are now the norm. Even if there have been some services directed to health community there are not many social bookmarking services created to this population segment. On the other hand the web development field has also evolved and many web applications framework have appeared implementing the concepts of the Model View Controller architecture. Finally the spam is still a problem affecting many web related services and the new social applications are not the exception.

Therefore the objective of this dissertation was to provide to the medical community an a Medical Social Bookmarking system developed in Python using the Django web application framework over the Google App Engine, to deploy the system in the appspot.com domain and in to analyse the process of developing an application in such framework and to determine the user behaviour interacting with the system. Apparently, this is the first time a medical social bookmarking services has been developed under those conditions.

# Table of Contents

|   |          |
|---|----------|
| <b>Chapter 1: Introduction</b>                              | <b>1</b> |
| <b>Introduction</b>   | <b>1</b> |
| 1.1 Project background                                      | 3        |
| 1.1.1 Spam Phenomenon                                       | 3        |
| 1.1.2 Web 2.0 Services                                      | 4        |
| 1.1.3 Spam in Web 2.0 Services                              | 5        |
| 1.2 Project aim and objective                               | 6        |
| 1.2.1 Project Aim   | 6        |
| 1.2.2 Project Objectives                                    | 6        |
| 1.3 Intellectual Challenge                                  | 7        |
| 1.3.1 Social Challenge                                      | 7        |
| 1.3.2 Technical Challenge                                   | 7        |
| 1.3.3 Scientific Challenge                                  | 7        |
| 1.3.4 Personal Challenge                                    | 8        |
| 1.4 Thesis Roadmap  | 8        |
| <b>Chapter 2: Web Applications and Web 2.0 Applications</b> | <b>9</b> |
| 2.1 Introduction  | 9        |
| 2.2 The World-Wide Web                                      | 10       |
| 2.2.1 Origins   | 10       |
| 2.2.2 Definition and Architecture                           | 11       |
| 2.2.3 Ajax  | 13       |
| 2.3 Web 2.0   | 14       |
| 2.3.1 Blogs   | 16       |
| 2.3.2 Micro-blogging  | 18       |
| 2.3.3 Wikis   | 19       |
| 2.3.4 Podcasting  | 22       |
| 2.3.5 RSS   | 23       |
| 2.3.6 Social Networks                                       | 25       |
| 2.4 Tagging   | 31       |
| 2.4.1 Folksonomy  | 31       |
| 2.4.2 Social Bookmarking Systems                            | 33       |
| 2.4.3 Tag Clouds  | 36       |
| 2.5 Conclusions   | 38       |

|  |           |
|--|-----------|
| <b>Chapter 3: Spam</b>                         | <b>40</b> |
| 3.1 Introduction                               | 40        |
| 3.2 Definition                                 | 41        |
| 3.3 Types of Spam and Detection                | 42        |
| 3.4 Conclusions                                | 49        |
| <b>Chapter 4: Software Development Process</b> | <b>50</b> |
| 4.1 Introduction                               | 50        |
| 4.2 Development Processes                      | 51        |
| 4.2.1 Waterfall Model                          | 51        |
| 4.2.2 Agile Framework                          | 53        |
| 4.2.3 Rational Unified Process (RUP)           | 55        |
| 4.3 Development Process Selected               | 58        |
| 4.4 Conclusion                                 | 59        |
| <b>Chapter 5: Analysis and Design</b>          | <b>60</b> |
| 5.1 Introduction                               | 60        |
| 5.2 High Level Analysis                        | 60        |
| 5.3 Architecture                               | 62        |
| 5.3.1 Model-View-Controller Architecture       | 62        |
| 5.3.2 Web Application Frameworks               | 63        |
| 5.3.3 Django                                   | 65        |
| 5.3.4 Google App Engine                        | 68        |
| 5.3.5 Architecture in Gray's 2.0               | 70        |
| 5.4 UML Diagrams                               | 71        |
| 5.4.1 Use Cases                                | 71        |
| 5.4.2 Class Diagrams                           | 79        |
| 5.4.3 Interaction Sequence Diagrams            | 80        |
| 5.5 Backend Design                             | 82        |
| 5.5.1 Entities and Relationships               | 82        |
| 5.5.2 Entity Relationship Diagrams             | 84        |
| 5.6 Conclusions                                | 85        |
| <b>Chapter 6: Development of Grays 2.0</b>     | <b>86</b> |
| 6.1 Introduction                               | 86        |
| 6.2 Development Environment                    | 87        |
| 6.2.1 Development Hardware                     | 87        |
| 6.2.2 Development Software                     | 88        |
| 6.2.3 Django Setup                             | 88        |
| 6.2.4 Google App Engine Setup                  | 91        |
| 6.3 Architecture                               | 93        |
| 6.3.1 Django on Google App Engine              | 93        |
| 6.3.2 Grays2.0 Architecture                    | 96        |
| 6.4 Development Process                        | 98        |
| 6.4.1 GUI Design and Template development      | 98        |

|                    |   |            |
|--------------------|---|------------|
| 6.4.2              | Handler Implementation . . . . .                        | 101        |
| 6.5                | Database . . . . .                                      | 115        |
| 6.5.1              | Data Model . . . . .                                    | 116        |
| 6.6                | Uploading Application . . . . .                         | 119        |
| 6.7                | Conclusions . . . . .                                   | 120        |
| <b>Chapter 7:</b>  | <b>Testing . . . . .</b>                                | <b>121</b> |
| 7.1                | Introduction . . . . .                                  | 121        |
| 7.2                | Blackbox / Whitebox . . . . .                           | 122        |
| 7.3                | IEEE 829 Testing . . . . .                              | 123        |
| 7.4                | Test Plan . . . . .                                     | 123        |
| 7.5                | Test Cases . . . . .                                    | 123        |
| 7.5.1              | Authorization . . . . .                                 | 124        |
| 7.5.2              | Adding or editing bookmarks . . . . .                   | 127        |
| 7.5.3              | Browsing or searching bookmarks . . . . .               | 130        |
| 7.6                | Conclusions . . . . .                                   | 134        |
| <b>Chapter 8:</b>  | <b>Evaluation . . . . .</b>                             | <b>136</b> |
| 8.1                | Introduction . . . . .                                  | 136        |
| 8.2                | International Evaluation Standards - ISO 9126 . . . . . | 138        |
| 8.3                | Evaluation Methodology . . . . .                        | 140        |
| 8.4                | Evaluation Results . . . . .                            | 142        |
| 8.4.1              | Users Profile . . . . .                                 | 143        |
| 8.4.2              | Adding bookmarks . . . . .                              | 144        |
| 8.4.3              | Searching of browsing for bookmarks . . . . .           | 145        |
| 8.4.4              | General Overview . . . . .                              | 146        |
| 8.5                | Conclusions . . . . .                                   | 148        |
| <b>Chapter 9:</b>  | <b>Conclusions and Future Work . . . . .</b>            | <b>150</b> |
| 9.1                | Introduction . . . . .                                  | 150        |
| 9.2                | Conclusions . . . . .                                   | 150        |
| 9.2.1              | Main conclusion . . . . .                               | 150        |
| 9.2.2              | Additional Conclusions . . . . .                        | 151        |
| 9.3                | Future Work . . . . .                                   | 155        |
| 9.3.1              | Medical Bookmarking System . . . . .                    | 155        |
| 9.3.2              | Other areas . . . . .                                   | 156        |
| 9.4                | Personal Remarks . . . . .                              | 157        |
| <b>Appendix A:</b> | <b>Grays2.0 Code Snippets . . . . .</b>                 | <b>158</b> |
| A.1                | Configuration Files . . . . .                           | 158        |
| A.1.1              | app.yaml . . . . .                                      | 158        |
| A.2                | App Engine Django Applicaton Handler Files . . . . .    | 158        |
| A.2.1              | views.py . . . . .                                      | 158        |
| A.2.2              | urls.py . . . . .                                       | 159        |
| A.2.3              | forms.py . . . . .                                      | 160        |

|   |  |            |
|---|--|------------|
| A.3   | App Engine Django Application Template Files . . . . . | 161        |
| A.3.1   | base.html . . . . .                                    | 161        |
| A.3.2   | header.html . . . . .                                  | 162        |
| A.3.3   | menu.html . . . . .                                    | 163        |
| A.3.4   | footer.html . . . . .                                  | 165        |
| A.3.5   | main_page.html . . . . .                               | 165        |
| <b>Appendix B: Evaluation Form . . . . .</b>          |  | <b>167</b> |
| <b>Appendix C: Data Analysis .py script . . . . .</b> |  | <b>168</b> |
| <b>References . . . . .</b>                           |  | <b>171</b> |



# List of Tables

|     |  |     |
|-----|--|-----|
| 2.1 | RSS Required channel elements. (RSS Advisory Board, 2009) . . . .                                | 24  |
| 2.2 | Top 20 Social Networking Websites. Accordingly to hitwise.com (week ending 04/11/2009) . . . . . | 28  |
| 2.3 | delicious most popular domains and tags. (Wetzker et al., 2008) . . .                            | 35  |
| 3.1 | Common patterns in Social Bookmarking Spam. (Wetzker et al., 2008)                               | 48  |
| 5.1 | User Entity . . . . .  | 83  |
| 5.2 | Bookmark Entity . . . . .  | 83  |
| 5.3 | Bookmark Entity . . . . .  | 84  |
| 6.1 | Hardware Specifications . . . . .  | 87  |
| 6.2 | Software Specifications . . . . .  | 89  |
| 6.3 | Google App Engine - Properties . . . . .   | 116 |

# List of Figures

|      |  |    |
|------|--|----|
| 2.1  | HTTP Protocol . . . . .  | 11 |
| 2.2  | Traditional vs Ajax application models. . . . .  | 13 |
| 2.3  | Web 2.0 Meme Map - O'Reilly 2005 . . . . .   | 16 |
| 2.4  | Twitter, Popular micro-blogging service . . . . .  | 19 |
| 2.5  | Editing entry in Wikipedia. . . . .  | 20 |
| 2.6  | RSS Feed Sample. <a href="http://media.acponline.org/feeds/annals.xml">http://media.acponline.org/feeds/annals.xml</a> . . . . . | 23 |
| 2.7  | RSS Aggregator: Google Reader. . . . .   | 24 |
| 2.8  | Linkedin, professional social networking site. . . . .   | 27 |
| 2.9  | Tag Cloud in del.icio.us . . . . .   | 37 |
| 3.1  | Monty Python's Flying Circus Spam Sketch. . . . .  | 41 |
| 3.2  | Spam E-mail Sample. . . . .  | 43 |
| 4.1  | Waterfall Model . . . . .  | 52 |
| 4.2  | Two dimensions of the Rational Unified Process . . . . .   | 56 |
| 5.1  | Model View Controller . . . . .  | 63 |
| 5.2  | Django Framework Functionality. . . . .  | 68 |
| 5.3  | Django Framework on the Google App Engine. . . . .   | 69 |
| 5.4  | Grays 2.0 on the Django Framework over the Google App Engine. . . . .  | 71 |
| 5.5  | Use Case Number1: Create a new account. . . . .  | 74 |
| 5.6  | Use Case Number2: Add a new bookmark. . . . .  | 76 |
| 5.7  | Use Case Number3: Search for bookmarks. . . . .  | 77 |
| 5.8  | Use Case Number4: Browse bookmarks by user or tag cloud. . . . .   | 78 |
| 5.9  | Use Case Number5: Delete Contents. . . . .   | 79 |
| 5.10 | Main Classes Diagram. . . . .  | 79 |
| 5.11 | Interaction Sequence1 - Add / Edit Bookmark. . . . .   | 81 |
| 5.12 | Interaction Sequence2 - Browse User Bookmark / Tag Bookmark. . . . .   | 82 |
| 5.13 | Entity Relationship Diagram - Draft1 . . . . .   | 84 |
| 5.14 | Entity Relationship Diagram for the Grays 2.0 Application (Final Version) . . . . .  | 84 |
| 6.1  | Django Testing Server - Welcome Screen . . . . .   | 91 |
| 6.2  | App Engine Hello World . . . . .   | 93 |
| 6.3  | Grays 2.0 Architecture . . . . .   | 98 |
| 6.4  | Generic Template . . . . .   | 99 |

|      |   |     |
|------|---|-----|
| 6.5  | Medical Bookmarks - Google Login . . . . .                                | 102 |
| 6.6  | Add Bookmark - Menu item on the green bar . . . . .                       | 107 |
| 6.7  | Add Bookmark - Error the user is not signed in. . . . .                   | 107 |
| 6.8  | Add Bookmark - The form to fill is printed . . . . .                      | 107 |
| 6.9  | Add Bookmark - Bookmark saved and redirection to the user's page. . . . . | 107 |
| 6.10 | Support Group - Search the datastore. . . . .                             | 119 |
| 7.1  | Registered user can browse own bookmarks. . . . .                         | 132 |
| 7.2  | User can browse bookmark by username. . . . .                             | 133 |
| 8.1  | ISO 9126 Standard . . . . .   | 138 |
| 8.2  | Use of social services to get feedback . . . . .                          | 142 |
| 8.3  | Familiarity with Web2.0 and Social Bookmarks . . . . .                    | 143 |
| 8.4  | Social services and type of user. . . . .                                 | 144 |
| 8.5  | Add a new bookmark . . . . .  | 145 |
| 8.6  | Add a new bookmark . . . . .  | 146 |
| 8.7  | Add a new bookmark . . . . .  | 146 |
| 8.8  | Appspot Dashboard CPU Usage. . . . .                                      | 147 |
| B.1  | Evaluation Form - Part1. . . . .  | 167 |
| B.2  | Evaluation Form - Part2. . . . .  | 167 |
| B.3  | Evaluation Form - Part3. . . . .  | 167 |

# Chapter 1

## Introduction

The World-Wide Web has been transformed in the last years with the apparition of social websites such as [wikipedia.org](http://wikipedia.org), [digg.co](http://digg.co), [del.icio.us](http://del.icio.us), and [flickr.com](http://flickr.com) changing the traditional content distribution model. In less than 10 years it has evolved and it has become a genuine extension of our daily life when the community social factors have begun to play an important role in the way we interact with the web. A simple Google search for the term “Web 2.0” returns almost 400 million hits linking to news, blogs and articles discussing this topic. These sites have changed from being a small subset of the web content with a specific user base, to one of the most important components of web sphere, posing a threat to the traditional and more established on-line and off-line businesses. Those types of sites are the soul of this new phenomenon that has been called “Web 2.0” by Tim O’Reilly(O’Reilly, 2005).

One of the main features of those social Web 2.0 sites is they rely on their users. In the traditional web sites used to provide static content generated by the publisher, but in the web 2.0 paradigm the owners of the websites depend on the user-generated content, they simply create the platform to publish and distribute this information. Those services create a symbiosis between the members and the

service and as soon as the user base grows and they use the service, it becomes better and better.

On the other hand healthcare is becoming more and more complex with thousands of articles published on a monthly basis, with new discoveries both in the diagnostic and therapeutic fields and with new challenging diseases tied to the developing world. Sterman discusses how difficult it is to generate reliable evidence through the scientific method and determine a valid hypothesis when there are an array of complex variables acting on the medical interventions or policies studied. He found that common mental model lead to erroneous and dangerous reassuring conclusions, that can persist for an significant amount of time that might hinder our ability to change (Sterman, 2006)

Nevertheless Bates found that one of the keys to improving safety will be improving access to reference information. There are already a wide range of textbooks, references on drugs, and tools for managing infectious diseases through access to the Medline database and thousands of clinical journals on-line (Bates & Gawande, 2003). One of the challenges facing health professionals is to cope with this overwhelming amount of medical information available on-line, since they need to be able to find it, to rank it and to prioritize it. One of the keys to solving this problem is to keep a bookmark list of the most relevant sites. Nevertheless a more useful approach would be to participate in a social bookmarking service since it would allow the sharing between the different users and everybody would benefit from the “wisdom of crowds” (Surowiecki, 2004). The downside of this incredibly dynamic scenario is that there is the temptation of spamming the service with misleading content or content that others user would not like to receive.

In Web 2.0 services, the methodologies and the spammer motivations change constantly and could manifest as creating phoney user profiles, filling the system with irrelevant or misleading annotations or corrupting the user-generated ranking

system. The spam phenomenon could threaten the existence of the social websites because spam can lower their relevance enough to discourage legitimate usage. This research aims to create a Medical Social Bookmarking Service entitled “Grays 2.0”, define and study the spam behaviour on the system and identify future measures to prevent and control the spam issue in a similar bookmarking system.

## 1.1 Project background

In this section an overview of the spam phenomenon is outlined, the Web 2.0 Social applications are reviewed, and the previous research on spam in social services is highlighted.

### 1.1.1 Spam Phenomenon

One of the risks that have to be taken into account in any web related service is spam. It is defined by the Merriam-Webster dictionary as “*unsolicited usually commercial e-mail sent to a large number of addresses*” and the anti-spam organization Spamhaus define it more precisely as an electronic message where the recipient’s identity and context are irrelevant because the message could be easily sent to a wide array of different recipients and this recipient has not granted clearly a permission for it to be sent (Glasner, 2001; spamhaus.org, 2008).

E-mail spam is probably the most common, the most problematic and the most studied type of spam. It can be defined as the bulk sending of unsolicited messages (normally with commercial intent). According to the statistics published by the e-mail security company Positini, spam activity has increased by 56 percent between September and October 2006. They also noted that almost 87 percent of the messages processed by companies were actually spam. Another study published by ipswitch in August 2006 found that on average 70 percent of the messages sent

to an email account were spam. Here they also analysed the main topics used on the messages and they found they were: Medications, Finance/Phishing, Pornographic, Electronics/Pirated Software and Mortgage Offers(postini.com, 2008; ip-switch.com, 2006).

E-mail spam can be annoying for the end user because it can flood their accounts with irrelevant messages and might even disable the Inbox, but it is also a huge problem for the Internet Service Providers, since they must assume the cost of dealing with the overwhelming amount of data transfer rates generated by these bulk messages. There is not only an economic cost in spam, but also an environmental impact too. In a study published by the anti-virus company McAfee (performed by ICF International and Richi Jennings), they found the energy cost of spam is huge: the energy spent in dealing with spam messages, transmitting and deleting them is comparable to the energy used by 2.4 million homes in the United States(Mcafee ICF International, 2009).

In summary, even if e-mail is the most common platform abused by spammers, in practice any communication media on the web is a target to be attacked with unsolicited messages. The economic factor is the main motivation of this activity: The cost of sending bulk messages is minimum and the return of investment can be huge since they might get a couple of customers or orders that will pay-off the continuous sending of unsolicited messages. The Internet and the World-Wide Web with their broad distribution and different array of services and media have opened a new window for the spammers to send the unsolicited material.

### **1.1.2 Web 2.0 Services**

The concept of “Web 2.0” was coined by Tim O’Reilly(O’Reilly, 2005). The term originated from a brainstorming session between O’Reilly and MediaLive International following a conference dealing with next-generation Web concepts and is-

sues. The discussion centred on the common traits in companies that had survived the dot.com bubble from early 2000. They perceived that there was a second generation of services on the web that could be considered a turning point into something different they called “Web 2.0”.

Web 2.0 focuses on dot companies thinking about services rather than package software, developing using open source methodology principles, trusting the users as co-developers, using the user-generated data as one of the competitive advantages and finally enabling user sharing and interaction through rich user interfaces. Another common features of those application are the use of tagging system (it has been called *folksonomy* in comparison with traditional taxonomy), the extensive use of open source solutions (as a LAMP stack combination) and the extension using APIs.

### 1.1.3 Spam in Web 2.0 Services

Folksonomy is one important component in many of these services such as `del.icio.us` and `flickr.com`. The tags that users add can be useful to find and share relevant resources and help to build and grow knowledge communities. Nevertheless the downside of the mounting popularity of those services is that they are beginning to become more susceptible to tag spam, defined by Koutrika *et al* as “*misleading tags that are generated in order to increase the visibility of some resources or simply to confuse users*”(Koutrika *et al.*, May 2007). In their study they attempted to simulate spam attacks to a tag based 2.0 sharing system. They found that the existence of popular tags provides many opportunities for malicious users to misuse tags and spam searches. They also found that the naive users are most vulnerable because they are more prone to search for unpopular tags where there will be more noise in the postings. One interesting phenomenon they noticed is that the spammers trying to attack the popular tags with their postings can have a smaller impact on



the system, since the community members have established a niche and query the system about tags created by their peers. They also proposed to have some kind of moderator for certain tags that might be targeted by spammers.

## **1.2 Project aim and objective**

### **1.2.1 Project Aim**

The main aim of this research is to identify the main web application frameworks and once one has been selected, to design and develop a functional Web 2.0 on-line Medical Social bookmarking service to provide an interesting and useful resource to the medical community. A secondary objective of the project will be to try to identify and avoid the underlying mechanisms on this collaborative system that prevents and favour the appearance of the spam.

### **1.2.2 Project Objectives**

- To plan and develop a Medical Social Bookmarking application and the corresponding dataset model, using Open Source Packages, to register a domain name and get a hosting service to publish the application and the database backend. Also to optimize the website and promote its visibility so it will be found by the search engines and the user base will grow.
- To offer the academic and professional Medical community a working useful Medical Bookmarking service to annotate and share interesting medical resources published on-line with their students and peers.
- To review the literature and find the most important measures to avoid the spam phenomenon in this Web 2.0 Social service.

## 1.3 Intellectual Challenge

In this section I will describe the intellectual challenge of my dissertation proposal on four levels: social, technical, scientific and personal level.

### 1.3.1 Social Challenge

As was noted by Bates (2003) in his paper the healthcare is becoming more and more complex with thousands of articles published constantly. Since one the keys to improving safety is to improve the access to reference information, this collaboration creates a new resource for the medical community where they will be able to annotate and share useful medical resources. The medical disciplines are knowledge intensive areas and therefore the proposed service can be relevant to them.

### 1.3.2 Technical Challenge

The technical challenge here is that it is not a trivial task to create a relevant Social Bookmarking service, using a web application framework, to optimise it and promote it in the search engines. It requires to master the Python programming language, to setup and test the framework and finally to deploy the application. A medical social bookmarking system in Django over the Google App Engine has not been attempted before.

### 1.3.3 Scientific Challenge

There are not many scientific articles dealing with the Web 2.0 Social Bookmarking systems and particularly in the Medical Bookmarking Systems, so this dissertation will contribute to the scientific literature on Web 2.0.

### 1.3.4 Personal Challenge

There are different areas why this topic is very appealing to me: I had the opportunity to assist at a Tim O'Reilly talk three years ago and since then I became interested in the Web 2.0 phenomenon. My original background is in Medicine but I have been working in web related companies during the last 5 years, so this project will give the opportunity to combine the knowledge acquired in both fields.

## 1.4 Thesis Roadmap

The rest of this dissertation will be organized in the following manner: Chapter 2 will contain a revision of the World-Wide Web, the Web 2.0 phenomenon, the origins of the term, the technologies associated with those services and the type of social services that are framed in the Web 2.0 term. In Chapter 3 a description of spam will be developed and the different types of spam will be analysed. In Chapter 4 the different software development process will be discussed and one particular methodology will be selected for the project. In Chapter 5 an analysis of the proposed social bookmarking system will be performed and the outcome will be a set of defined use cases, class diagrams and interaction sequence graphs modelling the main functionalities of the application. In Chapter 6 the actual development of the software will be presented, including the set-up of the development environment, the selection of the web application framework rationale and the actual implementation of the different cases. In Chapter 7 the different types of software testing methods will be discussed and the resulting product from the previous chapter will be tested and in Chapter 8 the evaluation of the product will be executed based on the feedback gathered from real users of the system. The dissertation will finish in Chapter 9 with the conclusions of the project.

# Chapter 2

## Web Applications and Web 2.0 Applications

*“Web 2.0 is the business revolution in the computer industry caused by the move to the Internet as platform, and an attempt to understand the rules for success on that new platform. Chief among those rules is this: Build applications that harness network effects to get better the more people use them. (This is what I’ve elsewhere called ”harnessing collective intelligence.”)!” (O’Reilly, 2006)*

### 2.1 Introduction

In this chapter the main features of traditional Web Applications and the of Web 2.0 Applications are compared and discussed. Since this dissertation is about a Medical Social Bookmarking system using the Web 2.0 technologies, the main components of Web 2.0 are relevant to this dissertation and they must be discussed in this chapter. A small overview of the World-Wide Web will be discussed and the main concepts of the Web 2.0 applications will be outlined. Also the main type of

technologies composing the services known as “Web 2.0” will be reviewed and their application on the medical and health field will be outlined. Finally the spam issue will be outlined, particularly in these kind of applications.

## **2.2 The World-Wide Web**

### **2.2.1 Origins**

The World-Wide Web was developed in 1989 at the CERN (European Organization for Nuclear Research) by the physicist Tim Berners-Lee. He wrote a proposal for managing the information by servers that could transmit document information through the Internet and linking the contents by using hypertext, using a simple but useful graphical interface. This idea was very well received by the physicists of the Institute and they began to use it to share the information gathered in the laboratory. `Info.cern.ch` was the address of the world’s first-ever web site and web server, running on a NeXT computer at CERN. In this site the visitors were able to learn about the project and create their own web page. During 1991 there were a additional web servers installed mainly in the United States and then there were already 300 servers running. The turning point was February 1993, when the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign developed the first version of Mosaic, a graphical browser to use on personal computers (Berners-Lee, 2008). In 15 years the number of unique URLs have grown in a exponential sequence. For example, Google passed from 26 million pages in its index from 1998 to one trillion 10 years later. (Alpert & Haja, 2008)

### 2.2.2 Definition and Architecture

The World-Wide Web (WWW) is the most popular and widely distributed system to access the Internet, in which items of interested such as Web pages, images, videos, games, etc, are shared using global identifiers called Uniform Resource Identifiers (URI). The World-Wide Web makes use of hypertext to access the different array of contents available on the world's different networks. This architecture allows people to publish content and it is easy to share by using hyperlinks. The World-Wide Web have evolved into an amazing information space of interlinked resources, across multiple languages, countries, cultures, and media. The web browsers such as Internet Explorer, Firefox or Safari are the software part used by the users to interact with the World-Wide Web. The main protocol used for exchange information in the web is the Hyper Text Transfer Protocol (HTTP) and most of the documents are encoded in the HyperText Markup Language (HTML)(Jacobs, 2004)

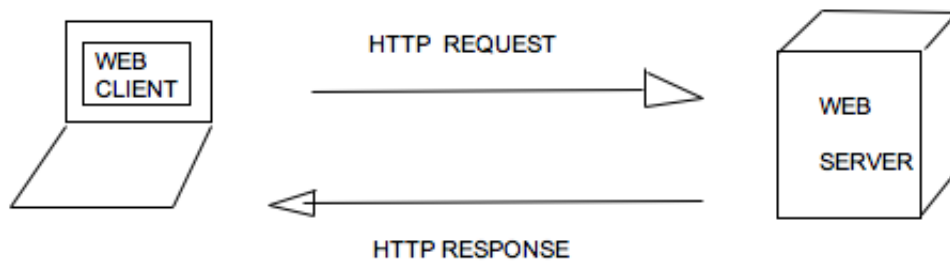


Figure 2.1: HTTP Protocol

In the web the URLs are used to identify and find resources. When a user enters in a web browser the URL of a resource, the browser sends a HTTP GET request to the server (The request by default goes via TCP/IP using the port 80). Once the web server receives this request, it will process the request and it will issue back an HTTP response specifying the type of content and the encoding of this content in

the HTTP Header. The user browser receives the reply and it knows how to interpret it based on content type defined on the header. The browser can issue additional HTTP requests to gather the additional elements of the resource (for example, it will need some additional requests to gather the images, CSS styles or JavaScript files). The resulting element displayed on the browser is commonly known as a “Web Page”. A typical http request made by a browsers such as Firefox might looks like this:

```
GET /exampledir/examplepage.html HTTP/1.1
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: en
```

And a typical response from a web server such as the open source Apache, might look like this:

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 09:23:24 GMT
Content-Length: 6821
Content-Type: text/html
```

HTTP uses the TCP / IP stack underneath as the transport protocol. When a HTTP request is made, it is necessary to establish a TCP connection with the server. Once this connection has been reached the server processes through their socket interfaces. The client sends the request into the socket and receives a response on it. The HTTP protocol does not need to care about the integrity of the request or the reply since those tasks are handled by the underlying TCP protocol. It is important to note the HTTP protocol does not keep information about the clients, so it is denominated as a stateless protocol (Berners-Lee et al., 1996) This type of

communications between web clients and web servers using the HTTP protocol, are the most important component of the World-Wide Web traffic.

### 2.2.3 Ajax

Ajax, is a collective term that includes the set of technologies enabling asynchronous communication between the web applications on the client side and the back-end application in the server side. JavaScript and XML have been on the web for several years but the ability to combine them to create dynamic applications has created a new wave of dynamic web sites. Using this approach it is possible to create rich web applications where the user do not have to wait until the content is downloaded from the server. Instead she is continually interacting with the application without noticing that the connection running in the background. The web applications in Ajax offer some behaviour that was only available before in desktop applications. It is able to overcome the main issues with the traditional web applications: the slow performance and the lack of interactivity. (Paulson, 2005; Garrett, 2005) The comparison between the main technological features of the traditional web applications versus the Ajax ones is displayed in the Figure 2.2.

Ajax is by definition interactive. In fact it is one of the advanced methods available on the web to develop rich and dynamic web applications that have gained a lot of popularity very quickly because it allows the page to update dynamically in small chunks without having to reload itself again. (Rahkila, 2006)

Ajax is popular because the user does not need to install additional plug-ins or extensions for browsing the content. The only requirement is to have a browser compatible with the latest web standards: JavaScript, XML, HTML and CSS. The Ajax application use an Ajax engine: a small service written in Javascript that using the XMLHttpRequest object is able to provide an asynchronous communication. Using Ajax the server can communicate constantly with the page calling it and this



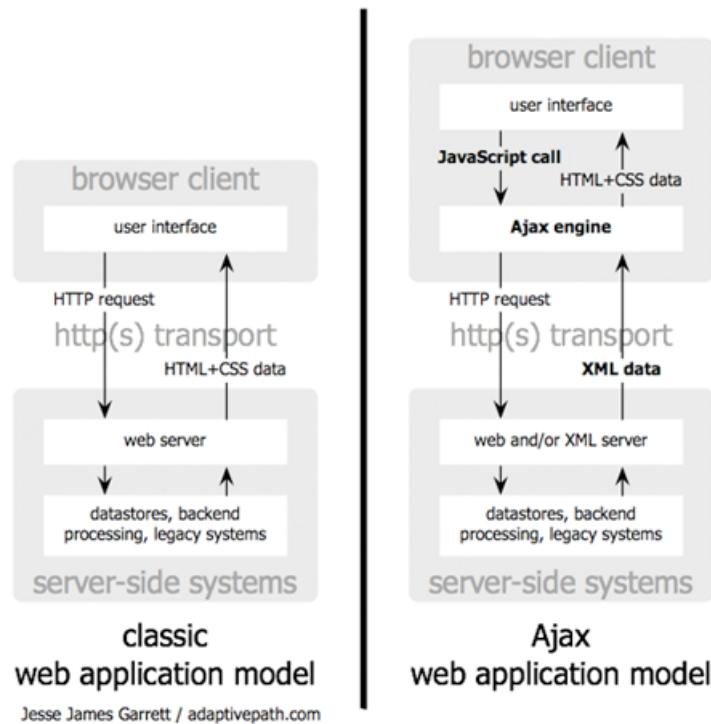


Figure 2.2: Traditional vs Ajax application models.  
(Garrett, 2005)

client can modify only the parts needed using a Javascript as well (Schools, 2009; McLellan, 2005). This model accelerates the process of displaying the information since the whole page does not need to be modified. The key elements in an Ajax application are:

- **XHTML and CSS** for presenting the content.
- **DOM** (Document object model) to modify and display the content.
- **XML** to interchange and manipulate the data.
- **XMLHttpRequest** object to get the data asynchronously.
- **Javascript** to integrate all these elements together.

There are quite a few Ajax applications running on-line, such as: Gmail, Google Maps, Calendar, Zoho Writer, Google Docs, etc. All of them are free and replace

many of the functionalities offered by their desktop software counterparts. There's one feature prevalent in most of these applications: The collaboration. Many users can share documents, edit them simultaneously, they can chat when they are reviewing their email or create a shared calendar where they can setup their meetings. Ajax applications allow a great deal of interaction between its users. Ajax is one the main components of many of the Web 2.0 applications that will be discussed in the next section.

## 2.3 Web 2.0

As was outlined in the introduction chapter, Web 2.0 is not a particular defined technology but rather made up by Tim O'Reilly, that originated from a brainstorming session between O'Reilly and MediaLive International following a conference dealing with next-generation Web concepts and issues. The discussion from this session was around the common characteristics of the web sites and services that were successful despite the dot.com bubble. They identified some common trends in those services that were clustered under the Web 2.0 umbrella. The points they identified as the most important features of this new paradigm were(O'Reilly, 2005):

- **Web as a Platform:** The web behaves like a platform, a kind of on-line operating system, where different sites offer services rather than just static content. These web services run in a similar fashion to their desktop counter parts.
- **Harnessing Collective Intelligence:** The links and the link structure generated by the humans are one of the most important factors in the determining relevance. The site is a medium where the users generate the content.
- **Data is the driving force:** The combination of licensed data plus the content generated by the users creates a distinct mesh-up difficult to recreate.

- **End of the software release cycle:** In the Web 2.0 companies don't think in terms of products but rather they work in terms of services that live in some sort of "perpetual beta" state where they are constantly monitoring, improving the service and releasing updates.
- **Lightweight Programming Model and Software Above Single Device:** The Web 2.0 creates web services developed to run in different devices in contrast to the traditional programming targeted to a single device.
- **Rich User Experience:** The web applications behave similar to the desktop applications but additionally they permit a high degree of collaboration. This collaboration is an integral part of those services.

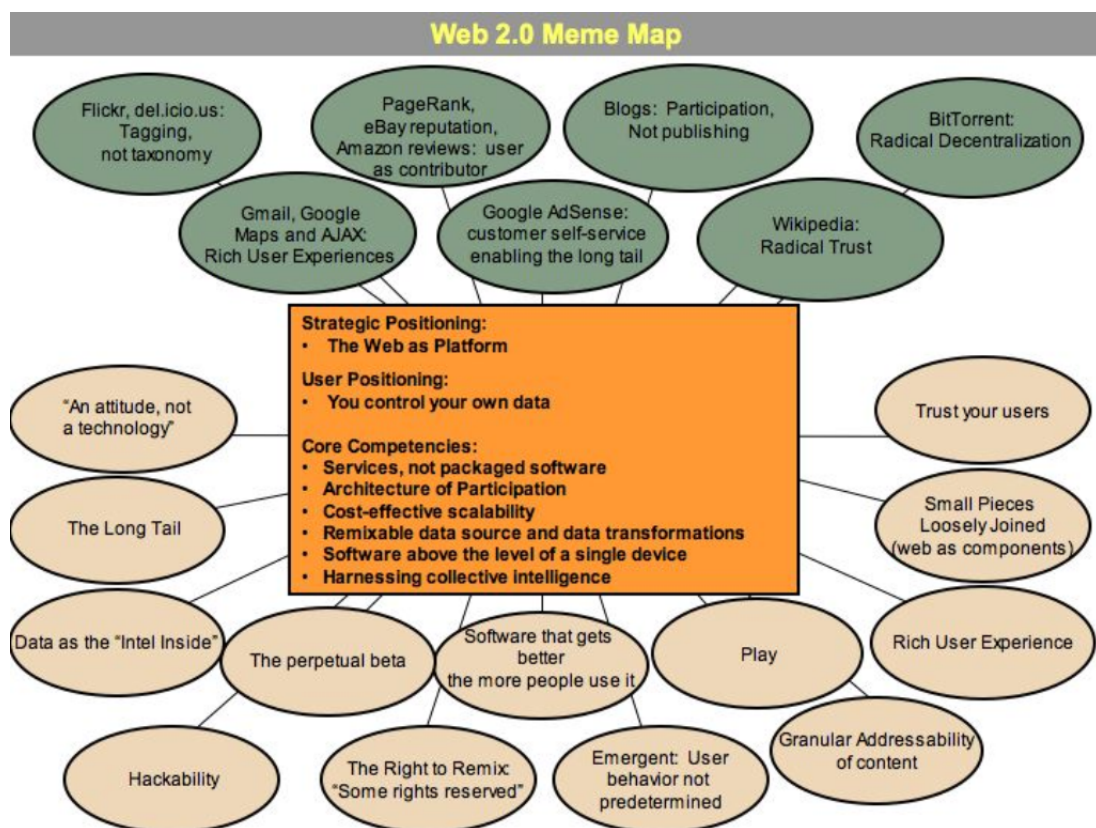


Figure 2.3: Web 2.0 Meme Map - O'Reilly 2005

In the Web 2.0 world companies think about on-line services, using open source methodologies, release their products on an agile methodology cycle but particularly rely on their users as collaborators and as an integral part of their content. In the Web 2.0 services domain the user feels empowered to create, modify and publish the content and share it with the rest of the community. Different technologies and applications categories such as wikis, blogs, RSS, folksonomy, social networks and tagging are a fundamental part of the Web 2.0 phenomenon and the following subsections will discuss them in more detail.

### **2.3.1 Blogs**

The word Blog is a contraction of Weblog and as its name implies it is a type of website or service where the owner update the contents on a regular basis with opinion entries, description of the life events or multimedia elements such as photos or videos. One of the very interesting features of these services is that the owners don't need to take care about the technical details of publishing and maintaining a site since the blogger tools take care of these issues for them. The word Blog can be used both as a noun or as verb, describing either the site where the entries are stored or the act of writing and publishing them (Horton, 2000). Another typical feature of blog services is that the entries are displayed in a reverse chronological order, so the last entries will be at the top of the page (it will be the first content seen by the visitors).

McLean et al. discussed how medical blogs are becoming more and more common and they include discussions, cases, diagnostic images or other topics that might be interesting for clinical practitioners. For example, Dean Giustini, a librarian at the University of British Columbia in Canada, maintains an interesting blog with reviews about topics useful for the medical students community (McLean et al., 2007). Given the fact that the web is becoming more and more of a multimedia plat-

form, blogging tools are becoming useful for sharing videos, audio files or images useful for the medical learning process. Actually since there are many phones web enabled with high resolution cameras embedded it is possible to get a high quality image from a clinical case and upload it in the fly to a blogging platform.

Another interesting feature of blogs is that they can be tracked using different technologies. One of the most useful ones are RSS (Really Simple Syndication). These are XML files created by the blog content systems that alerts its subscribers about the new entries published when they happen. Therefore the visitors interested in the content of a particular blog do not need to visit on a regular basis but rather go there when there is new material to be reviewed. Since the RSS are feeds that are sent to the browser, they can be loaded as live bookmarks or even be tracked with a feed aggregator such as Google Reader. There is even a new term coined in the web called “mashups” that involves the use of RSS feeds coming from different sources to create a new site aggregating them.

Most blogs have a particular niche of visitors following the topics discussed by the author, but it is common to have readers that at the same time are authors, and this particular interaction between the different bloggers generates one important community feature of those type of sites. Sometimes the entries are influenced by the writings of others and the basic linking nature of the web becomes an informal citation type between the different blogs. There are know bloggers in different areas and verticals and the interconnection between many of them has been named “blogosphere”.

There are two additional features of the blogging systems that helps the authors to interact with each other and follow the events happening in their peer blogs: *Blogrolls* and *Trackbacks*. The first one can be defined as a list of the other blogs that the author reads regularly and therefore she wants to endorse them by placing a link in her own. The sites that are linked in the blogroll normally write about

similar topics or they have contents interesting to the author linking to them. The trackbacks on the other hand are a more recent referencing feature of the blog systems to automatically update an entry with a reference to another blog hosted elsewhere, when the latter link to the former. If two sites have the trackbacks activated this automatic referencing system allows the author of an entry to discover if there is a discussion of his topics in places outside of his own blog (Marlow, 2004).

### 2.3.2 Micro-blogging

One the latest phenomenons in the Web 2.0 sphere that is beginning to develop but has grown exponentially during the last months is “Micro-blogging”. The principle of this new type of communication is very simple, the users of the Micro-blogging services update their friends or followers about their current status using small messages of usually less than 200 characters that are sent through the web or the mobile phones. This communication service has been popularised very quickly through services such as Twitter, Jaiku and Ponwce. Some Social Network services such as Facebook and the most popular instant messaging service like Yahoo, MSN and Gtalk also have a similar feature by offering a “Status” functionality (Java et al., 2007).

The most popular example of this type of service is Twitter that has become to Micro-blogging almost what Google is to web search. There are media companies using Twitter to send updates about their headlines like BBC or the New York Times and even in this difficult economic times some people is using Twitter API to develop applications to search for jobs. For example the company WorkDigital developed TwitterJobSearch, a job search engine using the twitter feeds (Gedda, 2009).

Also recently there was an Internet service outage for most of the users in the San Jose/Silicon Valley area of California, ATT turned to twitter to keep updated its customers updated about the the fiber cut causing the problems and the status of

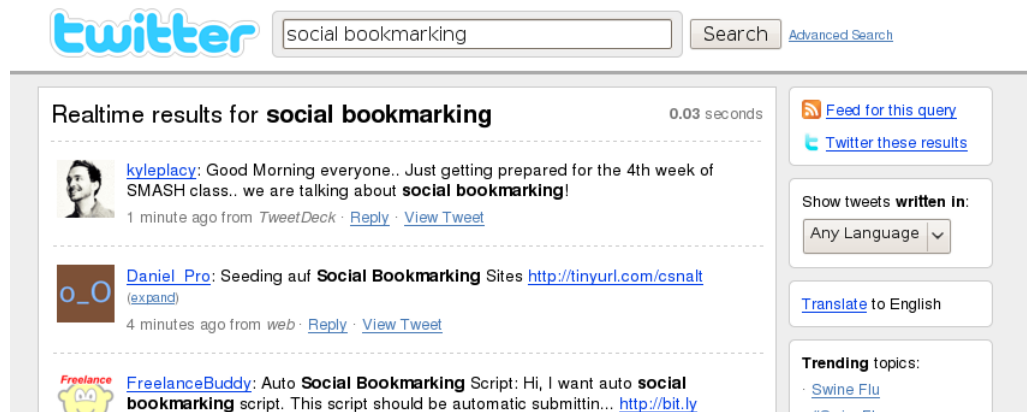


Figure 2.4: Twitter, Popular micro-blogging service

the service, with around 2400 following the feeds(Reardon, 2009).

In the medical world Micro-blogging can also be useful. Sidharth Kumar Sethi describes in his paper how Medical Doctors have many social interactions in the real world but they have little time to manage them, so Micro-blogging services such as twitter presents themselves as an effective system to keep in contact with the different social circles like friends, patients and colleges. Some of the potential uses he describes are Twitter pages for the Doctors working in a hospital so the nurses and patients can track them, the conferences where the organizers could update the participants on the events and announcements and some special channels for tracking difficult patients or review specific conditions(Sethi, 2009).

Even if Micro-blogging is one the newest services of the Web 2.0 is growing very quickly and there are more users and institutions getting to use those services in many different creative ways. It seems there is a lot potential to grow and probably there will be appear in the future new applications and use cases for the Micro-blogging phenomenon(Java et al., 2007).

### 2.3.3 Wikis

The most well-known wiki system is *Wikipedia*, an on-line encyclopaedia where any user can add or edit any content to the site creating a vast repository of the human knowledge. This type of collaborative tool where users can fully edit the content of the website themselves was created by Ward Cunningham, who got decided to name the tools after the Hawaiian term “wiki” meaning quick or to hurry. In wiki systems the users can browse the different contents of the website and they can edit them freely. They can also reorganize the contents and even the structure of the site. The only tools a user needs to create content in a wiki are a web browser and an Internet connection. The open nature of those systems position them as a very powerful part of the Web 2.0 phenomenon, creating a platform to create and share this user-generated knowledge and also to keep it up to date (Augar et al., 2004).

One interesting element about the wiki phenomenon is the community element. Wikis not only allow the users to generate knowledge but also to collaborate between them, a very important feature. The contents are not published by a single expert but it is rather the result of the work by different authors using the wiki system. Wikis normally include a versioning system allowing the users to revert to a previous version if there are disagreements with the last changes (Boulos et al., 2006).

One of the paradoxes of wiki systems is their open nature where any user can modify any content. This is one of its most important features because it allows a real collaboration among the wiki users but at the same also a known drawback since anyone can modify the contents and sometimes it can compromise the quality of the contents and lead to vandalism. A vandal might edit the valid contents of a good page, add inappropriate texts or even delete all the content of a page. Wikis also can trigger a phenomenon known as *edit wars* when different people have an



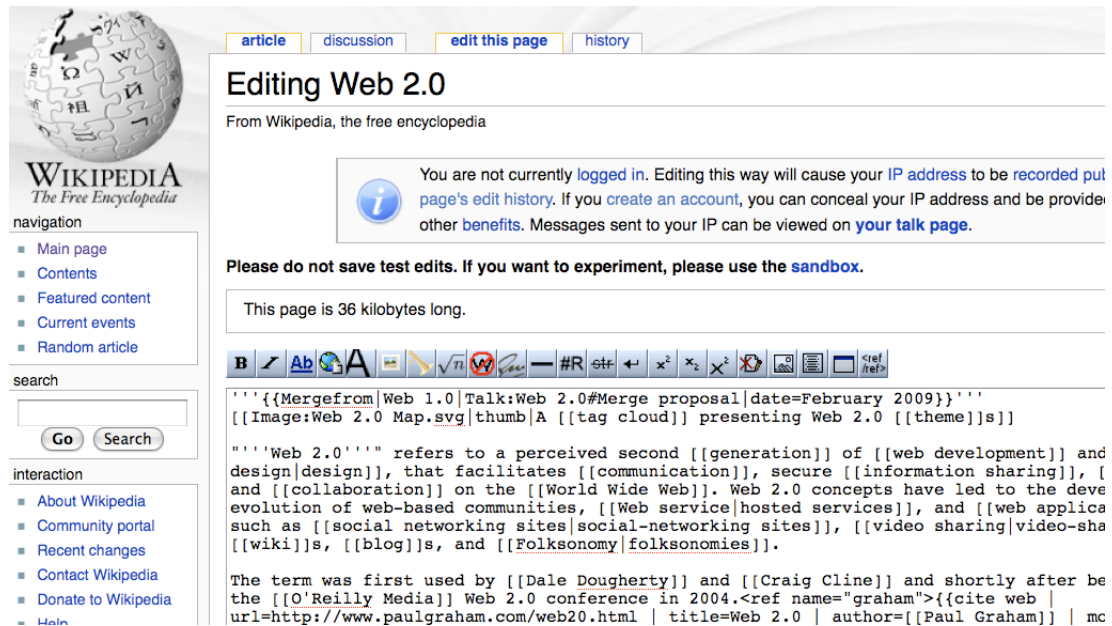


Figure 2.5: Editing entry in Wikipedia.

opposite opinion about a particular topic and they edit and revert the pages over and over in order to highlight their point of view. A known case of this edit wars happened with the page of the American ex-president George W. Bush. Since some of the users were editing this page trying to highlight his failures and at the same time some other some others viewed him more favourably. In the end both parties reached a consensus having a version as much objective as possible and the article was semi-locked by the administrators (Brain, 2005).

Another potential issue with wikis open nature is that anyone can copy and paste copyrighted material without the authorization of the author. Finally another issue related to academic standards of those contents is that wiki systems promote community creation of contents and not individual authoring. Since most of the contents are created by a group of people in different time frames and authorial identification is very hard and therefore wiki contents are normally not considered as valid academic references. Nevertheless, despite all these drawbacks when mistakes in wiki contents are compared with more traditional and recognized reference

materials, surprisingly there are no big differences between them (Giles, 2005).

In the medical field there are some additional factors to take into consideration since health records, medical past, diagnostic images and test results are sensitive patient information and must be dealt with care. It is necessary for the authors of health wikis to get permission from patients before posting that kind of information on-line and also it is important to anonymize it. Nevertheless there have been some successful cases of health-related wikis such as Ganfyd, an on-line wiki style medical reference website edited by medical professionals and non-medical experts (Boulos et al., 2006).

In conclusion wiki systems are one the most interesting products from the Web 2.0 era, that created a framework for truly sharing the human knowledge, with an amazing capacity to keep it up to date by the minute. There are risks related to the quality and the bias in the contents on those pages, but the community itself eliminates the errors in the text of those pages by the subsequent revisions. Therefore wikis presents as an interesting reference system for medical knowledge.

### 2.3.4 Podcasting

On October 23, 2001, Apple released an unexpected product, the iPod, a mp3 music player that instead of using flash memory, used a small hard drive disk, and since it uses a very intuitive set of controls through a scroll-wheel the product became an instant success and a created a whole “ecosystem” around the product, with sound systems, accessories and music stores targeting those devices (Darling, 2006). One of the positive consequences of the popularity of those iPods and the competing mp3 players is that it has created created an new expression media in the websphere: Podcasts. A podcast is defined by the new oxford dictionary as “*a digital recording of a radio broadcast or similar program, made available on the Internet for downloading to a personal audio player*” and by Merriam Webster as “*a program (as of music or*

*talk) made available in digital format for automatic download over the Internet". This term combined the terms of "Broadcast" and "iPod" to encompass all those digital files in mp3, wma, aac, etc created by individuals or institutions and made available to download automatically using scripts, to play off-line on the road when wanted.*

In the medical field there are some very useful podcasts available to both health professionals and patients. Dr. Des Dimov maintains in his medical blog a list of the health themed podcasts he listens on a regular basis to keep up to date with the medical news, discoveries and advancements. Some of the most trusted sources of medical institutions such as the American College of Cardiology and the prestigious Johns Hopkins University offer medical updates through podcasts and some of the most read medical publications such as the Journal of American Medical Association, the New England Journal of Medicine, the Annals of Internal Medicine and the British Journal of Medicine offer summaries about the contents published in the magazines using the the podcast format (Dimov, 2006). Normally podcasts are hosted in a web server and are updated on regular basis and the user can subscribe to the feeds of the podcast so they receive an automatic update on their feed readers every time a new podcasts has been published. Those feeds normally use RSS (Really Simple Syndication). An RSS sample of the Annals of Internal Medicine can be found in the figure 2.6

### **2.3.5 RSS**

Really Simple Syndication (RSS) is a popular set of formats created for syndicate content easily through feeds which the reader subscribes to, for getting automatic updates. There are different types of websites and media using this format but in general the common factor is that they are content providers updating frequently their sites such as news websites, blogs, magazines and podcasts. The RSS feeds are defined using the XML format (extensible markup language), a generic tag based

```

<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0" xmlns:itunes="http://www.itunes.com/dtds/podcast-1.0.dtd" xmlns:atom="http://www.w3.org/2005/Atom">
  <channel>
    <title>Annals of Internal Medicine Podcast</title>
    <link>http://media.acponline.org/feeds/annals.xml</link>
    <description>Issue highlights and interviews from Annals of Internal Medicine.</description>
    <copyright>Copyright 2007 American College of Physicians</copyright>
    <lastBuildDate>Tue, 7 Apr 2009 10:12:00 EST</lastBuildDate>
    <itunes:explicit>no</itunes:explicit>
    <itunes:author>American College of Physicians</itunes:author>
    <language>en-us</language>
    <itunes:image href="http://media.acponline.org/feeds/images/annalspodcast.jpg"/>
    <itunes:owner>
      <itunes:name>American College of Physicians</itunes:name>
      <itunes:email>podcast@acponline.org</itunes:email>
    </itunes:owner>
    <itunes:category text="Science & Medicine">
      <itunes:category text="Medicine"/>
    </itunes:category>

    <atom:link href="http://media.acponline.org/feeds/annals.xml" rel="self" type="application/rss+xml" />

    <item>
      <title>Smoking</title>
      <link>http://www.annals.org/content/voll50/issue7/images/data/DC1/annals_20090407.mp3</link>
      <description>New approaches to smoking cessation; interview with Nancy Rigotti, MD, of Harvard Medical School; plus
      <pubDate>Tue, 07 Apr 2009 10:29:08 EST</pubDate>
      <guid>http://www.annals.org/content/voll50/issue7/images/data/DC1/annals_20090407.mp3</guid>
      <itunes:duration>00:26:52</itunes:duration>
    </item>
  </channel>
</rss>

```

Figure 2.6: RSS Feed Sample. <http://media.acponline.org/feeds/annals.xml>

language used in many different applications for storing and sharing data. The user can “Subscribe” to one of this RSS feeds by getting the link to the XML file and storing it either in browsers such as Mozilla Firefox as a dynamic “Live Bookmark” that updates automatically based on the contents of the feed, or add it to certain “RSS readers or aggregators” applications or websites where is possible to store, organize and read all the feeds in one single place (one of them is shown in the Figure 2.7).

The creator of the first version of this format (RSS 0.91) was Dan Libby, who was working for Netscape Communications and created this new XML type of format to define a My Netscape Network Channel and keep the reader updated on the changes published on these personal sites. Nevertheless the company stopped supporting the format and another couple of developer groups: Dave Winer from UserLand and the RSS-DEV Working Group keep developing RSS independently (Cadenhead & Smith, 2006). The current version of the RSS format is 2.0, that defines what is RSS: “RSS is a Web content syndication format. Its name is an acronym for Really Simple Syndication. RSS is a dialect of XML. All RSS files must conform to the XML

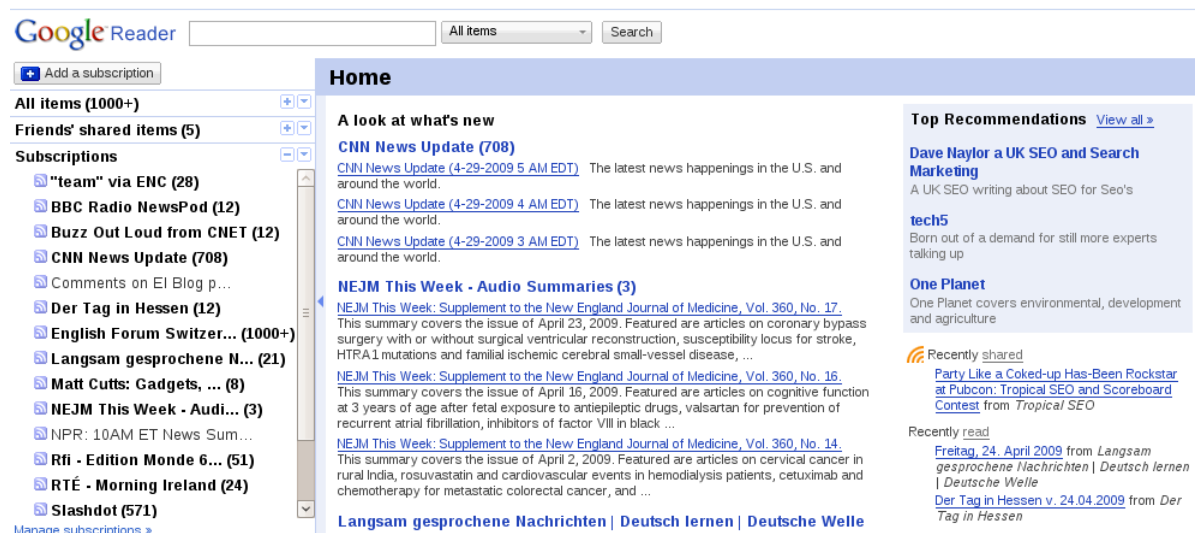


Figure 2.7: RSS Aggregator: Google Reader.

1.0 specification, as published on the World-Wide Web Consortium (W3C) website.” and specifies which ones are the header and the required and optional fields the file must contain in order to be a valid RSS document. The required fields are summarised in the Table 2.1.

In the medical field RSS feeds are very useful to health professionals since they do not need to visit their favorite websites on a regular basis, but instead they can subscribe to the appropriate RSS feeds and receive automatic alerts once the sites have been updated with new content. Since now there are the Internet enabled smartphones such as the iPhone and the Blackberry are very popular and many doctors own one of these or a similar product, they can receive updates about the medical news or cases irrespective of where they are.

Giustini discusses how many doctors are using RSS as a new method of discovering new information since they have realised the traditional methods of finding information such as search engines or the traditional Medline are not efficient enough to find the latest and more relevant information available about a particular

| Element     | Description   | Example   |
|-------------|---|---|
| title       | The name of the channel. It's how people refer to your service. If you have an HTML website that contains the same information as your RSS file, the title of your channel should be the same as the title of your website. | example.com News Headlines                                  |
| link        | The URL to the HTML website corresponding to the channel.   | http://www.example.com/                                     |
| description | Phrase or sentence describing the channel.  | The latest news from example.com, a Sample Journal Website. |

Table 2.1: RSS Required channel elements. (RSS Advisory Board, 2009)

topic. These busy health professionals are using RSS readers to fight information overload and aggregate only the sources they are particularly interested in such as the evidence based medicine Cochrane Library (Giustini, 2006).

On the other hand Barsky in his paper describes the kind of RSS feeds that can be useful for Health Librarians. He recognised that Blog updates, the newspapers and journals articles, press releases from health organizations and medical database updates using RSS as the main type of feeds useful for the Health Professionals working in libraries and documenting centres (Barsky, 2006).

In conclusion RSS is a very useful feed format that allows content publishers to update its followers on the fly when new materials are published by using these XML compliant files. On the other end the RSS files allow the end users to subscribe to feeds of their favourite sites and aggregate all the information in one single application or website where they can discover quickly when new content has been published.

### 2.3.6 Social Networks

Nowadays the term social network is associated with popular on-line Social Network systems. Nevertheless the origin of this concept does not come from the Internet or the World-Wide Web world but it is rather a concept described in the anthropology sphere. The term “Social Network” was used first by the Professor J. A. Barnes, a Reader in Anthropology in the University of London, who studying in Bremnes, a small fishing village in Norway, discovered that the whole social life could be described in terms of points linked by lines forming in practical a term a set of complex relationships between the members of that society and creating indeed a network structure (Barnes, 1954). Even if this concept was interesting it was not particularly known world-wide, but once with the Web 2.0 movement began to develop and several social network services began to emerge such as MySpace and Facebook, the concept became very popular and actually and today some Social Websites are some of the most visited on the World-Wide Web.

Boyd and Ellison defined in their paper a social network as “*web-based services that allow individuals to construct a public or semi-public profile within a bounded system, articulate a list of other users with whom they share a connection, and view and traverse their list of connections and those made by others within the system.*” The majority of social networks permit its users to create an account where they can define information they want to share and they are able to construct their networks step-by-step by adding new connections to people they already know off-line or by creating new bonds with the new member they meet on-line. One interesting finding of their paper is that the main objective of social networks is not about meeting strangers but rather to articulate existing social circles and unveil their social networks. It is possible also to make connections with people that would be very hard using an off-line scenario, but for most users Social Networks are a medium to communicate with people they already know and that belong to their

existing social networks (Boyd & Ellison, 2007).

Even if many of the popular social network sites are general, so they are used by many users for many purposes, most of them had at the beginning a “Thematic” origin: For example Facebook was originally designed to keep in contact with the college students, MySpace was mainly a place to share music influences or politic thinking and Friendster was a place to start romantic relationships. Nevertheless some social sites are still focused mainly in one particular segment like LinkedIn, which focuses in the work environment, Last.fm that is mainly about music or Couchsurfing that is focused on the hospitality and the traveller niche.

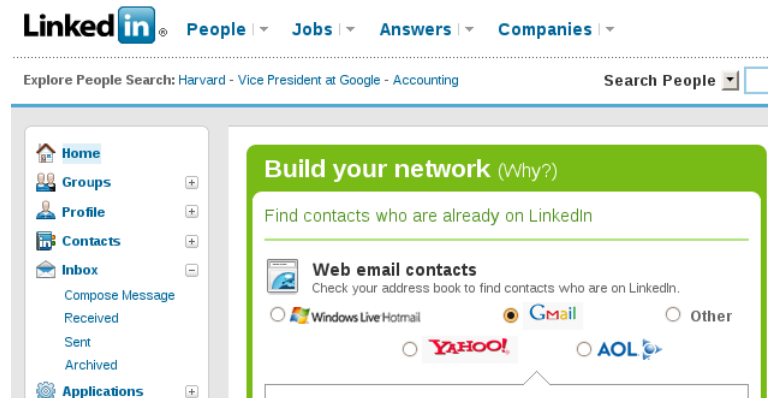


Figure 2.8: LinkedIn, professional social networking site.

The social network services have experienced an important growth recently and since many of them have embedded email and instant messaging services, they have become one of the most commonly used communication media to keep in contact with friends and acquaintances.

There are a broad array of social network services on-line, for example the Open Directory project lists in the social networking category more than 180 different websites of this type (dmoz.org, 2009). Some of the most popular services in the category “Computers and Internet - Social Networking and Forums” reported by the American Internet traffic monitor company are listed in the Table 2.1.

Some of the websites listed are not exclusively Social Network services but rather



| Rank | Property             | Domain              | Visits |
|------|----------------------|---------------------|--------|
| 1.   | MySpace              | myspace.com/        | 31.37% |
| 2.   | Facebook             | facebook.com/       | 29.84% |
| 3.   | YouTube              | youtube.com/        | 8.67%  |
| 4.   | Tagged               | tagged.com/         | 1.72%  |
| 5.   | Yahoo Answers        | answers.yahoo.com/  | 1.41%  |
| 6.   | myYearbook           | myyearbook.com/     | 1.01%  |
| 7.   | Twitter              | twitter.com/        | 0.95%  |
| 8.   | Yahoo Groups         | groups.yahoo.com/   | 0.94%  |
| 9.   | Windows Live Home    | home.live.com/      | 0.59%  |
| 10.  | Classmates           | classmates.com/     | 0.48%  |
| 11.  | Yahoo Profiles       | profiles.yahoo.com/ | 0.45%  |
| 12.  | Meebo                | meebo.com/          | 0.45%  |
| 13.  | MySpaceTV            | vids.myspace.com/   | 0.41%  |
| 14.  | Bebo                 | bebo.com/           | 0.38%  |
| 15.  | Yahoo Message Boards | messages.yahoo.com/ | 0.37%  |
| 16.  | BlackPlanet.com      | blackplanet.com/    | 0.33%  |
| 17.  | LinkedIn             | linkedin.com        | 0.31%  |
| 18.  | hi5                  | hi5.com/            | 0.31%  |
| 19.  | Google Groups        | groups.google.com/  | 0.31%  |
| 20.  | Mylife               | mylife.com/         | 0.29%  |

Table 2.2: Top 20 Social Networking Websites. Accordingly to hitwise.com (week ending 04/11/2009)

services with some social components. If we extract Social Network-only services we can conclude that the most popular ones in the United States are MySpace (31.37%), Facebook (29.84%) and Tagged (1.72%). Nevertheless in certain European markets Facebook has become very popular such as the British one where it has surpassed MySpace. One report published by the British Office of Communications in 2008 found that in the leader in the UK market for the Social Networks site was Facebook, followed by MySpace and Bebo (Ofcomp, 2008). An interesting fact related to the Facebook growth is that a report published just one year earlier by the information provider Comscore, described how Bebo was the most visited Social Network site in the UK (Also this report describes the astonishing growth of the British traffic to the property Facebook.com, since it had grow in 366% in only 6 months (comscore, 2007).)

In the health field Social Networks are begining to show their potential in giving support to patients with a particular condition and to discover the effectiveness of new treatments to control certain diseases. For example, social network Patients-LikeMe offers to its users the possibility to connect with some other users with similar conditions and share experiences about symptoms, treatments, prognosis, etc. The site is divided into channels and the patients can join big groups for the main neurological, immune and mood diseases. It is interesting that using the same concepts as traditional social networks the patients are able to share information about the treatments they are receiving and discuss any technique that can improve their outcome. For example Aldhous, mentions in his paper how some community of users suffering from amyotrophic lateral sclerosis (ALS), have joined forces to try out a new therapy using Lithium, to control their disease (Aldhous, 2008).

Dr. Luo also discussed how even if health professionals are not a particularly technical knowledgeable population and there are only a limited number of professional Social Networking in the medicine field, still they are beginning to get into

the Social Network phenomenon. MyMedwork is one of those Social Health Sites where physicians, biomedical engineers, psychologists, pharmacologists and public health researchers can gather in a Social Network environment, maintain contacts with their peers, find and contact again their former classmates, colleagues or collaborators, or creating and developing new relationships across the globe. This site also allows its participants to tag their publications with traditional PubMed tags, so it is easier for them to look for similar topics in Medline (Luo, 2007).

As I have reviewed during this section the multiple advantages of the Social Networks services such as the ability to reveal and keep alive existing social connections and extends the social circles, by having a profile in a social service and share information with friends or acquaintances, but one of the disadvantages discussed by Gross and Acquisti is the privacy issues. They studied the patterns of information revelation in on-line social networks, particularly in Facebook, and they found that the large majority of the studied population are careless about sharing their personal information and the limiting privacy preferences are rarely used. They believe a consequence of this behaviour is that the users are at risk of attacks on their personal and on-line presence and they might be victims of stalking and identity theft. Another risk is that some users might create a dossier of the personal information available and sell to a third party. In conclusion they found that while personal data is generously shared, the limiting privacy features are hardly used, putting the user at risk (Gross & Acquisti, 2005). This issue is also discussed by Aldhous in the case on PatientsLikeMe portal, since the information shared the patients can be identified by potential employers or insurance companies and it will have an adverse outcome on them (Aldhous, 2008).

In conclusion recently Social Network services have evolved and developed enormously during the last years and they have become one of the most discussed topics in the news related to Web 2.0 services and sites like Facebook have demon-

strated the potential of these kind of services, even in the Medical Fields. Nevertheless privacy is a factor that will have to be discussed as well, since the growth of social networks services also increase the risks.

## 2.4 Tagging

### 2.4.1 Folksonomy

The term Folksonomy was coined by Thomas Vander Wal in a mailing list discussion related to information architecture, when he was describing the classification systems being developed in popular services such as Delicious and Flickr. In his view this term was a combination of *folk* (the community) and *Taxonomy* (the classification), so it is a classification system where the users of the systems define themselves the categories, through tagging systems and where unlike the formal taxonomies schemes, there is not a formal set of hierarchies, but rather a flat system defined by this community of users. (Smith, 2004)

Mathes (2004) analyses the relationship between the traditional methods for generating metadata taxonomy systems and the new folksonomy classification in the Web 2.0 applications. He describes how metadata is normally a set of structured information describing the main characteristics of different types of materials such as articles, books, etc. The problem he found is that this metadata is normally expensive to gather and to maintain, since it requires professionals trained in Library Information Sciences using a strict set of guidelines to make sure the quality and accuracy of the classification is preserved. When there is a vast amount of information generated quickly this methodology is not suitable since the information processing takes time and the professionals don't have the resources to scale the process, particularly in media with an exponential growth such as the World-Wide Web. A second alternative he describes is author-generated metadata, where the

creators of the documents add information about the documents they are creating. Even if this method is more scalable than the first one it has the disadvantage that the users are not involved in the process so each author acts in their own island of information.

Finally he discusses a third methodology where the users are the main actors involved in the creation of this metadata. Folksonomy can be included in this category. The social bookmarking service was one of the first one of its kind in using this, Folksonomy was used to define the classification of websites. Here the users are able to add their favourite sites and they are tagged with a set of keywords named “tags”. Something interesting about this model is that the bookmarks are public, so users are not only saving their favourite sites but also sharing them with the rest of the community. In this model the same websites are tagged with different keywords and it creates a user-based classification system. Some of the disadvantages found by the author are the ambiguity and the limitation of the categorization system: It is difficult to deal with synonyms and also it is challenging to set-up the boundaries of the tag usage in the system, to avoid overlapping (Mathes, 2004).

Actually McLean *et al* acknowledge that this might be one of the issues for the Health professionals used to search in professional medical databases such as MEDLINE where the headings and subheadings are clearly specified. They think that the lack of a standard hierarchy might prove to be challenging to those users and they might criticise the possibility of using polysemy, synonyms and plurals, as it is the case in many of the Web 2.0 folksonomy based applications (McLean et al., 2007).

In another paper related to the folksonomy categorization, Hotho and Jaschke, add to the discussion a new term: they define the term “personomy” as the set of resources a user add to the system tagging them with a particular keyword or set of keywords and they consider the folksonomy as a collection of those personomies. In many of those systems they found that a particular user can see all the elements

he has added to the system but once he browse one of them he can get additional info provided by the other users of the system (Hotho et al., 2006a).

In conclusion folksonomies are a new classification methodology that was born on the World-Wide Web, where the users of a system collaboratively creating and managing tags to annotate one particular kind of content, generate the categories and therefore the classification categories. The main difference with traditional methods is that there is not an expert involved in the taxonomy, but rather many user classification based on their personal criteria and the combination of all those personal opinions define a collective knowledge. Even if there are risks regarding the ambiguity, overlapping and accuracy using this method, particularly in the Medical field, it is one of the most useful scalable methodologies in the World Wide Web huge arena.

### **2.4.2 Social Bookmarking Systems**

The bookmarks concept is probably as old as the World-Wide Web itself, since Mosaic, the first browser developed by the National Center for Supercomputing Applications offered the possibility to save their favourite websites using a feature called “hotlists”. Once Netscape Navigator, the Mosaic successor was developed, the feature was incorporated in the form of bookmarks and also when Microsoft developed their own version of the web browser they also included this functionality, through favourites. Bookmarks were very useful in the initial stages of the web to keep a list of the sites the user wanted to visit again, but the problem inherent with the system was that personal bookmarks used to grow very quickly and were very hard to maintain. Search engines are one of the best solutions to this issue because they were crawling and indexing the web all of the time, so in a way they offered a dynamic bookmarking service where the user could find on the fly sites they have visited before just by entering keywords, so people get used of having their favourite

sites at hand, just a search query away (Hammond et al., 2005). Nevertheless with the development of Web 2.0 concepts where collective knowledge and collaboration are the foundation of many services, the concept of storing the recommended sites begun to emerge again but with the additional community element, it was not only to go back again to the favourite ones, but also to share them with the rest of the community adding even an editorial component on them. *Del.icio.us* was the first one of these Social Bookmarking services where the users could store their favourite sites and share them with the community, by “Tagging” the resources using the keywords that are considered appropriate for each one of them. As was discussed before these new features are called “Folksonomy” by Thomas Vander Wal, when he was describing the users ability to classify the resources based on their own criteria and how the sum of those personal opinions were creating a user bottom-up generation of metadata (Smith, 2004). *Del.icio.us* defines social bookmarking as a service designed to allow its users to store and to share bookmarks on the web, instead of inside the browser. It identifies three main advantages of the social bookmarking versus the traditional bookmarking (delicious.com, 2007):

- The users can get to the bookmarks from anywhere, no matter where they are (athome, at work, in a library, or on a friend’s computer).
- It is possible to share the bookmarks publicly, so friends, coworkers, and other people can use them for reference, amusement or collaboration.
- The people can find some other people in Delicious who have interesting bookmarks and they can add them to their own collection of links.

*Del.icio.us* allows its users to gather in a central place their bookmarks and share them if desired with any web visitor. To add a bookmark the user needs to be logged into the system (it is necessary to register, but the process is quick and free) and

| Position | Domain  | Tag      |
|----------|---|----------|
| 1.       | <a href="http://en.wikipedia.org">http://en.wikipedia.org</a> | design   |
| 2.       | <a href="http://www.youtube.com">http://www.youtube.com</a>   | blog     |
| 3.       | <a href="http://www.flickr.com">http://www.flickr.com</a>     | software |
| 4.       | <a href="http://www.nytimes.com">http://www.nytimes.com</a>   | web      |
| 5.       | <a href="http://www.google.com">http://www.google.com</a>     | tools    |

Table 2.3: delicious most popular domains and tags. (Wetzker et al., 2008)

must provide the URL, the title of the bookmark, notes (before it was called extended description) and any tags the user consider relevant to that resource. The system tries to grab the title from the live page and offers some tag suggestions (popular tags and recommended tags), so the bookmarking process is even more streamlined. If the user can also install many browser plug-ins that integrates the social bookmarking system into the normal browser bookmark behaviour. One interesting feature is that user can check how the other users of the systems have saved a bookmark (for example what title have they chosen and what tags they have used to associate the resource).

In a paper published in 2008 by Wetzker, Zimmermann and Bauckhage, they found the popularity of the service has grown exponentially since it was released in 2005, when they measured variables such as bookmarks, users, tags and URLs per month. In their study they created a dataset consisting in 142 million bookmarks downloaded from the service during a 4-month period starting in September 2007. They started their web crawling by getting the bookmarks associated with the tag “web2.0” and found the other tags associated with it and keep doing this process recursively during the stated period. Their analysis on this data is very interesting since the top bookmarked domains and the top selected tags are related to the technology domain(Wetzker et al., 2008). The top five domains and tags are displayed in the Table 2.3.



Del.icio.us is the most known social bookmarking service and it has the broadest audience but Hammond describes some other services with a similar concept. For example CiteULike that is a bookmarking tool for academic links (the users can bookmark papers from certain websites that have academic content, and from which the site has been able to collect citation metadata), Connotea is a citation manager that retrieves metadata from sites like PubMed, HubMed, Amazon.com or Nature.com. FURL is another service that besides saving the bookmark stores a copy of the page contents (Hammond et al., 2005).

Social bookmarking is used in the medical field as a means of share references between practitioners and researchers. For example researchers can tag their favourite academic papers so practitioners working on the same field can find additional useful references (Varlamis & Apostolakis, 2007). Barsky also discusses how these tools are an excellent way of discovering new resources when the researchers are looking for a particular topic, since they can dig for analogous materials, that some other users have bookmarked using similar tags (Barsky & Purdon, 2006)

In conclusion Social Bookmarking is one the first activities associated with Web2.0 but at the same time represents some of the main principles of the computing movement: *Harnessing Collective Intelligence*. Bookmarking was one of the first activities related to World-Wide Web browsing and Social Bookmarking is an extension of this task with a collaboration layer on top that allows the user not only to keep their favourite sites but also to share them with the world and vice-versa.

### 2.4.3 Tag Clouds

As was discussed in the Web 2.0 and the folksonomies subsection, one of the main features of Web 2.0 applications is to harvesting collective intelligence by allowing the users to generate and categorize the content. One of the common ways of classifying the content in one of these applications is by using Tags, which are a set

keywords or term assigned to a piece of information that contains metadata, that means information about the resource being tagged. Popular bookmarking websites like del.icio.us, the video portal youtube and the photosharing service flickr use tags to classify and categorize the resources on those systems. Once the services become popular and the users actively tag most of their contents one the challenges that emerge is how effectively display the tags so they can quickly come back to the tagged resources. One solution implemented by many websites is Tag Clouds, defined as *“a set of words, typically a set of tags, in which attributes of the text such as size, weight or colour can be used to represent features (e.g., frequency) of the associated terms.”* (Halvey & Keane, 2007).

One of the first services to use Tag Clouds was Flickr, but the concept was also popularised by the social bookmarking service Del.icio.us and the blog search engine Technorati. Those services uses the most common type of Tag Cloud implementation where the size and position of a unique tag represents the number of items in the systems that have been identified with that particular tag. An example of a Tag Cloud that is offered in the social bookmarking service delicious can be observed in the Figure 2.9

Even if the Tag Clouds are not a particularly intuitive method of navigation they can provide to the visitors and users an instant snapshot of the resources contained into the whole service or in a subset of it. The Tag Cloud represents a summary of the content of the website and the main topics are highlighted by size, position or colour. These clouds are not considered as a replacement of the traditional navigational methods but rather as alternative method. In fact even if Tag Clouds are one of the typical design elements associated with the Web 2.0 web applications and many websites have tried to implement them to a certain extent, sometimes they have created some unfriendly and unintuitive patterns that don't accomplish their objective (Smashing Magazine, 2007).

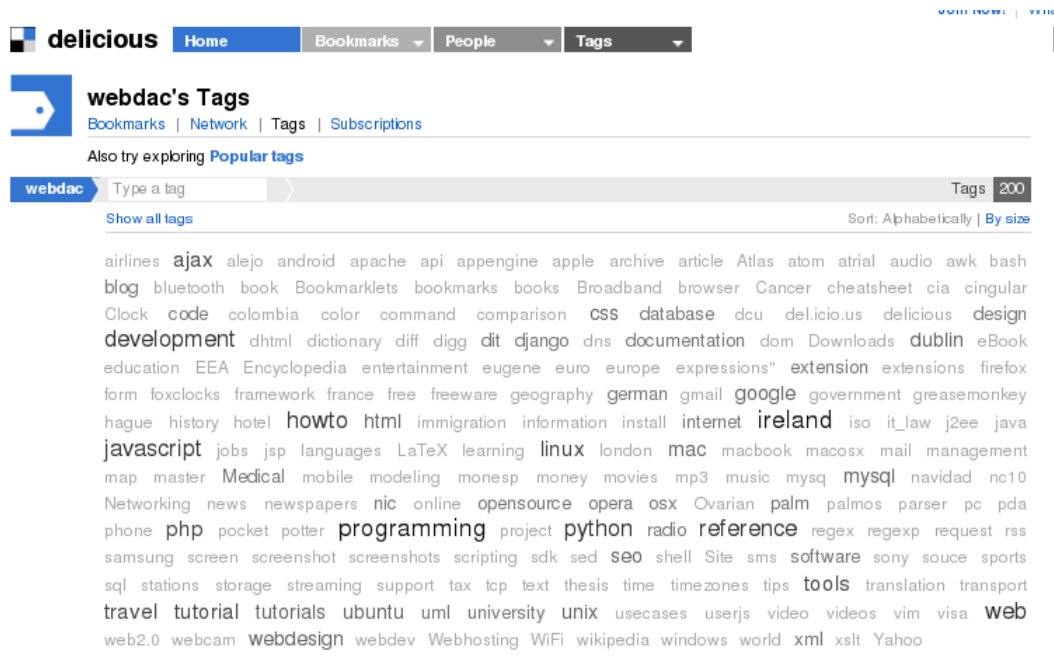


Figure 2.9: Tag Cloud in del.icio.us

Actually Halvey and Keane studied the relationship between how the tags were presented in a tag cloud and how easy it is was to find information. Some of the interesting findings of this study are that the alphabetization and the font size are very important and they determine the interval between presenting the cloud, scanning the list and finding the information (Halvey & Keane, 2007).

Tag clouds are also used in some of the Web 2.0 medical applications. For example the Medical Blogs RSS aggregator medworm offers a Tag Cloud section where the most popular blogs in the MedWorm database tagged by topic are displayed using this method. Here they highlight the importance of a particular Tag by size, shading and color and one interesting option they offer is the possibility of building the cloud by week, month or year; on the other hand it is also possible to rearrange the cloud and convert it into a storm, where the most popular tags appear on top and the least popular on the bottom, or into lists where the tags are categorized alphabetically or by popularity (medworm.com, 2009).

In summary even if the Tag Clouds are not perfect and are certainly not a re-

placement for the traditional navigational methods, but they are properly used they can give to the users a first glimpse of the most popular resources stored in a Web 2.0 application.

## **2.5 Conclusions**

Since this dissertation is about a Medical Social Bookmarking system using the Web 2.0 technologies the main purpose of this chapter was to review traditional web applications versus the new Web 2.0 ones. This new paradigm in web development was discussed and the main technologies such as Ajax were discussed. Also the main services composing Web 2.0 (blogs, RSS, micro-blogging, podcasting, social networks, folksonomy and tagging) were studied in the detail and the realm of medical applications on those domains was explored.

After doing this literature review it is clear that the Web 2.0 is not a marketing plot as sometimes it has been called but rather a real evolution of the World-Wide Web applications using different components of technology such as Ajax. A useful finding of this review is that even if the search engines are very popular for finding information on the web, there is an important amount of users that still are actively bookmarking their favourite sites by using Social Bookmarking services so they can share the resources with other users. Also it was useful to realise that the medical community is beginning to get into the social web and use the main Social Networks, RSS aggregators, medical blogs, etc, but there is room for improvement and it seems a Social Bookmarking system for the health segment like the one proposed in this dissertation could be useful for them.

# Chapter 3

## Spam

*“Under no circumstances will I ever purchase anything offered to me as the result of an unsolicited e-mail message. Nor will I forward chain letters, petitions, mass mailings, or virus warnings to large numbers of others. This is my contribution to the survival of the online community (Roger Ebert, The Boulder Pledge )” (Royce, 1970)*

### 3.1 Introduction

In the previous chapter the features and services of Web 2.0 Applications were compared and discussed and one of the conclusions from that review is that there is an important number of user using blogs, RSS, micro-blogging, podcasting, social networks and also social bookmarking systems. One of the risks of having a successful product on the web is the possibility of abuse by users who will use the service as a platform to promote a third party site of products. In this chapter the spam phenomenon will be discussed, understanding what can be defined as spam, how it can affect Web 2.0 services and how it can be contained.

## 3.2 Definition

The term spam is defined by the Merriam Webster dictionary as *"unsolicited usually commercial e-mail sent to a large number of addresses"* and the anti-spam organization Spamhaus define it more precisely as an electronic message where the recipient's identity and context are irrelevant because the message could be easily sent to a wide array of different recipients and this recipient has not granted clearly a permission for it to be sent. The origin of the word to denominate this behaviour can be traced to the episode 25th of the British television series Monty Python's Flying Circus where the customers are trying to get a dish from breakfast and everything contains spam (a canned meat product) and at certain moment the word spam overrides the dialogue and the spam takes over the Scene and it is impossible to hear anything but spam, so the word becomes really annoying .(Glasner, 2001; spamhaus.org, 2008).



Figure 3.1: Monty Python's Flying Circus Spam Sketch.  
From [youtube.com/watch?v=ODshB09FQ8w](https://www.youtube.com/watch?v=ODshB09FQ8w)

The first spam message registered on the Internet was sent by Digital Equipment Corporation in 1978. Gary Thuerk, an aggressive DEC sales employee thought it would be interesting to inform all the Arpanet users that the company had integrated the Arpanet protocol support directly in one of their new machines. Even

if the message could have been relevant for the users there was a negative reaction and the ARPA administrator lodged a formal complaint. Later on, before the bulk messaging began to be problems in the email systems, Usenet, a world-wide distributed discussion system divided in a set of "newsgroups" with names that are classified hierarchically by subject, experienced the phenomenon: Several groups began to receive messages asking for money (charity messages) and *make money fast* postings. Nevertheless this issue was not identified as a problem and neither labeled as such until 1994 when a religious sect send a message warning that Jesus was coming soon to every single Usenet group, Canter and Siegel, a pair of lawyers working in the immigration field also did a posting to all the groups advertising their services to help the recipients to participate in the green card lottery and it provoked an angry reaction from the Usenet users which decided to retaliate and to flood their Inbox and fax numbers with phony messages. From that moment the spam term became popular and users began to refer to those types of communication as spam Templeton (2001).

### 3.3 Types of Spam and Detection

Traditionally the term spam have been used in the e-mail realm as the sending of unsolicited bulk messages using the SMTP servers. Even if e-mail is the most common platform abused by the spammers, in practice any communication media on the web is a target to be attacked with unsolicited messages. The economic factor is the main motivation of this activity: the cost of sending bulk messages are minimum and the return on investment can be huge since they might get a couple of customers or orders that will pay-off the continuous sending of unsolicited messages. Brian Marshall have a small demonstration in his article about how spam works, where he explains how if you spam a 100 friends with a message for offering

a 5 box order and you get 2 orders is a great deal since the investment was almost non-existent and there was return. If this small experiment is run in a greater scale, the profits can augment exponentially and that is the reason why the spam continues to be attractive for many people (Brain, 2003).

The Internet and the World-Wide Web with their broad distribution and different array of services and media open a new window for the spammers to send the unsolicited material. As was discussed in the previous section, newsgroups are another favourite target for the spam, and other web mediums to be attacked are the web search engines, blog service providers, wikis and ultimately mobile phone text messaging systems have also be affected by spam.

### **E-Mail Spam**

E-mail spam is probably the most common, the most problematic and most studied type of spam. It can be defined as the bulk sending of unsolicited messages (normally with commercial intent). According to the statistics published by the e-mail security company Positini, spam activity has increased in a 56 percent between September and October 2006. They also noted that almost 87 percent of the messages processed by the company were actually spam. Another study published by ipswitch in August 2006 found that in average 70 percent of the messages sent to an email account were spam. Here they also analysed the main topics used on the spam messages and they found they were: Medications, Finance/Phishing, Pornographic, Electronics/Pirated Software and Mortgage Offers (positini.com, 2008; ipswitch.com, 2006). Laurent Oudot wrote an interesting article in security focus summarising the modus operandi of the E-mail spammers (Oudot, 2003). These findings were also highlighted in a can-spam report to the US congress in 2005:

- **Email Harvesting:** In this phase the spammers try to build a database with an updated list of their targets. They use different mechanisms for gathering



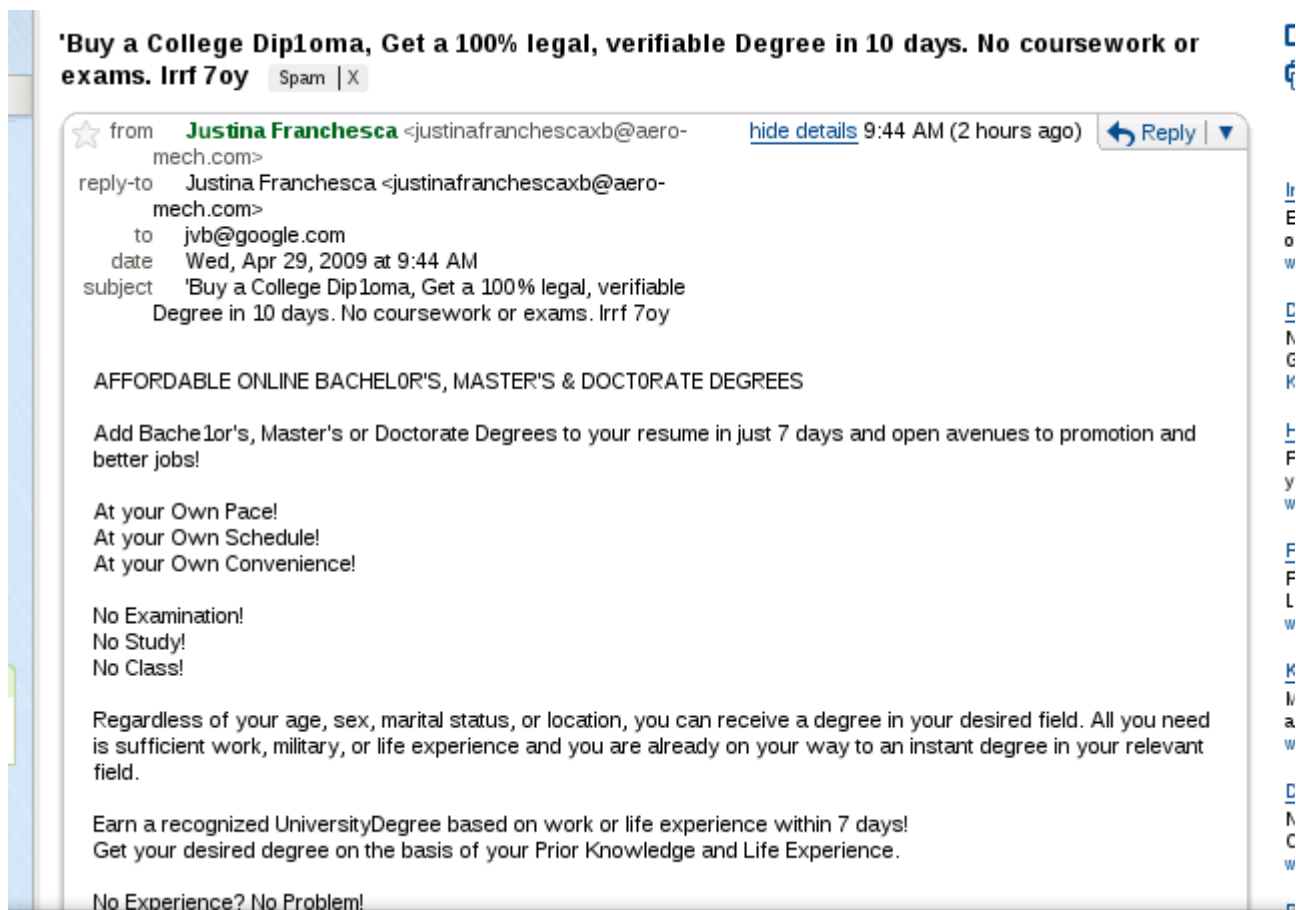


Figure 3.2: Spam E-mail Sample.

the addresses. They get them from the messages in Usenet, by creating robots that scan all the messages and try to find any text next to the (from and reply-to) fields. They also get them from mailing list revealing the recipients and finally they can create web crawlers which surf the web and collect all the links containing the the mailto parameter.

- **Open Proxies:** This is another useful tool in the spammer arsenal. Instead of connecting directly to a remote mail server they use an open proxy. This service allows the spammer to send the messages passing through a middle-man server. The main advantage for the spammer is that they can keep their anonymity. The mail server logs will show the proxy IP address instead of the original sender who made the requests. Sometimes they even chain their

request with several open proxies trying to hide their identity even more.

- **Open Relays:** Open relays are defined as a Mail Transfer Agent that receives messages from third party sources even if they are not the destination, and it is able to route them to another third party. Since these relays forward messages that they have not posted to or by a local user, they are abused by the spammers. They use this service to send bulk messages to random email addresses. Once the spam passes through one of these open relays the header changes and it appears as it was originated there.
- **Zombie Drones:** This is the latest trend used by some malicious spammers to distribute unsolicited messages. Instead of using open proxies or open relays, they use viruses, worms and Trojans that hijack the home user computers and turn them into a open proxy "zombie" they can control to send the spam. This is a serious problem for the users because they might not be aware of this problem until the ISP closes their account.

E-mail spam is annoying for the final user because it can flood their mailboxes with irrelevant messages and might even disable the inbox, but it is also a huge problem for the Internet Service Providers, since they must assume the cost of dealing with the overwhelming amount of data transfer rates generated by those bulk messages.

### **Web Search Engine Spam**

One important difference between the e-mail spam the web-based spamming is the inherent technology tied to the HTTP protocol on the World-Wide Web. Spammers can not send their messages directly to the users because the model works through a request / response model, therefore the users need to ask for the web resource before it is delivered. The best way for spammers to circumvent this limitation is

by the abuse of the Search Engines given the fact that they are the primary source of websites when web users are looking for information. Spam in this medium is done by deceiving the search engines and compromising the good will relationship between the engine and the user (Castillo et al., 2006). Spamdexing (search engine spamming) is defined by Gyongyi and Garcia-Molina as “*any deliberate action that is meant to trigger an unjustifiably favourable relevance or importance for some Web page, considering the page’s true value*”. It was also defined by Perkins as “*Any attempt to deceive a search engine’s relevancy algorithm*”. He introduces an interesting concept: anything that would still be done if search engines did not exist or anything that a search engine has clearly given written permission to is excluded from the Web Spam definition.

We can summarise search engine spam as the use of any mechanism to artificially raise the ranking of a site a web page in the search engine results pages. In order to achieve this objective the spammers try to understand the main factors determining the search engine ranking system and try to trick the algorithms in order to take advantage of the flaws. There are different techniques used to achieve those results and different authors have tried to classify them in separate groups. One of the first attempts to classify the techniques was done by Perkins who divided Search Engine Spam in two categories: Content spam and Meta spam. On the other hand Gyongyi and Garcia-Molina in their Webspam Taxonomy paper divide the techniques in two big categories: Boosting Techniques (Including Term spamming and Link spamming) and Hiding Techniques (Including content hiding, cloaking and redirection) (García-Molina & Gyöngyi, May 2005; Perkins, 2001). Finally Baoning in his dissertation tries to unify the different spam attempts in the following three categories: (Wu, 2007)

- **Content spam:** It is about changing the content of the page in order to rank higher. It includes mechanisms such as repeating the keywords several times

on the document, adding additional query keywords that are not related with the content of the page or abusing of the *title*, *meta* or *alt* tags by filling them with keywords.

- **Link Spam:** The point here is to modify the link structure of the page and the anchor of the links to try to achieve a better ranking. This is achieved by participating in link farms (Each page has a collection of links that point to almost every other page), engaging in link exchange schemes, link buying or by buying expired domains inheriting the page rank of the previous domain.
- **Page-hiding Spam:** This technique is about presenting a different page to the search engine, usually with more content. This is achieved by IP and user agent based cloaking (presenting a different version to the web crawlers), redirects (the spam page is seen by the robot but not by the user) and by layer based spam (hiding part of the page using CSS styles.)

### **Blog Spam (Splogs)**

As was discussed previously blogs are web sites composed of dated entries typically listed in reverse chronological order. They have become an influential phenomenon on the web and an interesting media to follow the events in the different niches. Therefore spam have been also to appear on the blogosphere manifested through spam blogs, called also splogs, created with the sole purpose of making money by publishing advertisements or affiliate content, normally using contents auto-generated or copied from other legitimate sources.

Blog spam is seriously affecting the blogging community since they blog search services are getting overloaded searching through these splogs and the main hosting platforms are under attack by this new trend of spam. Kolari et al. and Mayfield conducted an study with the objective to identify and classify the blog

spam phenomenon. Based on the works of Gyongyi and Garcia-Molina on the web-spam taxonomy they created their own classification adapted to the blogosphere. The categories they proposed are: *non-blogs*, *keyword stuffing*, *post-stitching*, *post-plagiarism*, *post-weaving*, *link-spam* and other techniques. In summary they found that the splogs are created with text that is automatically generated or plagiarized, are made for publishing just advertising or affiliation content or can be created just to inflate the page rank of other domains(Kolari et al., 2006b).

### **Social Bookmarking Spam**

Attempts to artificially raise the importance or ranking of a webpage do not target only search engines, but can affect any user-generated content website like the social bookmarking ones. Wetzker et al., analysing a big corpus data coming from the del.icio.us social bookmarking service they noted a high occurrence of entries posted by automatic mechanisms that were considered spam with a high impact. One surprising piece data is that the 19 out of the 20 most active users of the system belonged to this spam category. Some of these users were tagging the same resource thousand of URLs in the same domain with only a few tags (Wetzker et al., 2008).

When they analysed the patterns of spam entries in the system they found some common features that can be useful for implementing spam detection measures in future Social Bookmarking systems. Some of the common characteristics discovered in that study are summarised in the Table 3.1

In a recent study published by Heyden, Koutrika and Garcia-Molina(2007) they outlined the mechanisms they used detect and fight the spam phenomenon: Identification-based techniques where both the users and moderators manually can flag certain

| Pattern               | Description   |
|-----------------------|---|
| Very high activity    | Automated posting with a higher rate than the human users |
| Few domains           | The URLs posted belong to a small set of domains          |
| High Tagging Rate     | Bookmarks tagged with an enormous amount of tags          |
| Very Low Tagging Rate | High number of entries with low amount of tags            |
| Bulk Posts            | To enter many bookmarks in bulk                           |

Table 3.1: Common patterns in Social Bookmarking Spam. (Wetzker et al., 2008)

contents as spam and also automatic detection based on patterns such as the IP addresses, the user name patterns, the referring domains, or text tags used. Also the sharing patterns were useful to determine if some sources or bookmarks were behaving unnaturally and could be suspicious of being spam.

The second strategy they used was a rank-based approach where the content likely to be spam was demoted and sent to the bottom of the results so even if the spam was still there, it was not apparent to the legitimate users. Finally they implemented a prevention interface model where they limit the opportunities to malicious users to spam the system. The authors found two ways of accomplish this goal; On one hand they set up CAPTCHAs to prevent automated account creation or automated bookmark posting and in the other hand they also set up a system to hide or reduce the value of certain interfaces by using personalization methods on a per-user basis, so it would be extremely difficult to the spammer to compromise the bookmark list of the bulk users. Finally they evaluated other alternatives such as making users who register for an account or post a bookmark to either pay a small fee or compute a proof-of-work. Nevertheless they pointed out that those interventions could scare off legitimate users away. One interesting parameter discussed in this paper is Spam Metrics. They take into account each object in the system (the URL in a Social Bookmarking system) and define spam as a malicious behaviour of

selecting an incorrect tag for a particular object or adding irrelevant object to the system. They conclude that the traditional spam fighting techniques might work better in Social systems than in the email-based scenarios because the users have to register but they acknowledge that spam techniques might evolve and rende them obsolete(Heymann et al., 2007).

## 3.4 Conclusions

The objective of this chapter was to review the origins and main characteristics of the spam phenomenon particularly for the potential abuse of the Web 2.0 applications that might appear. Even if e-mail spam is the most known type, the conclusion of this review is that the problem can appear in any service where the user can interact with the system. The splogs and fake accounts in the Social Bookmarking systems are a serious issue that can compromise the quality of the service. It is fair to assume that as Web2.0 services have become popular they will be abused by spammers and therefore some spam detection and prevention is required.

# Chapter 4

## Software Development Process

*“For some reason what a software design is going to do is subject to wide interpretation even after previous agreement. It is important to involve the customer in a formal way so that he has committed himself at earlier points before final delivery. To give the contractor free rein between requirement definition and operation is inviting trouble” (Royce, 1970)*

### 4.1 Introduction

In the previous chapter the main characteristics of new Web 2.0 Applications were compared and a review of the most important services and technologies of the Web 2.0 sphere were described. Also the concepts of Folksonomy, Tagging and Social Bookmarking were highlighted and the possibility of implementing one of those services in the health domain was discussed. The objective of this project is to implement a Social Bookmarking system so after having done a review of the research relevant to this topic the next step is to discuss the different software development process and identify the most appropriate one for this project.

The most important software processes are discussed in this chapter and the reasoning about why a particular process is better than the others for this project



will be explained.

## 4.2 Development Processes

A software development process can be defined as the methodology or structure set by an organization or individual that is used for developing or maintaining a software project (Pes, 2009). There are alternatives to managing the development process but all them try to describe the different tasks and time related to software development including requirements gathering and analysis, functional specifications, software architecture, design, design, implementation, testing, deployment and maintenance. The main objective of the process is to find a set of systematic best practices that can be repeated over and over in order to keep a high quality and good efficiency in the software development process.

Some of the processes use a sequential approach where every step is a prerequisite to the next one and there are also some iterative methodologies where the phases are smaller and in the cycles, and the different phases are revisited. The most well known software development process models that will be discussed next are: Waterfall Model, Agile Software Development and Rational Unified Process.

### 4.2.1 Waterfall Model

The waterfall model is a software development process sequential by nature where each of the steps have to be completed rigorously before going to the next one and the process follows a steady flow passing from analysis, design, implementation, testing, integration and maintenance; so that the start of each of these stages must await the completion of the one immediately before. The analogy to a waterfall comes from the idea that the flow of the process should go head down and in theory it should not be possible to pass to the next stage before the current one has

been completed. The main principles of this model are presented in the Figure 4.1

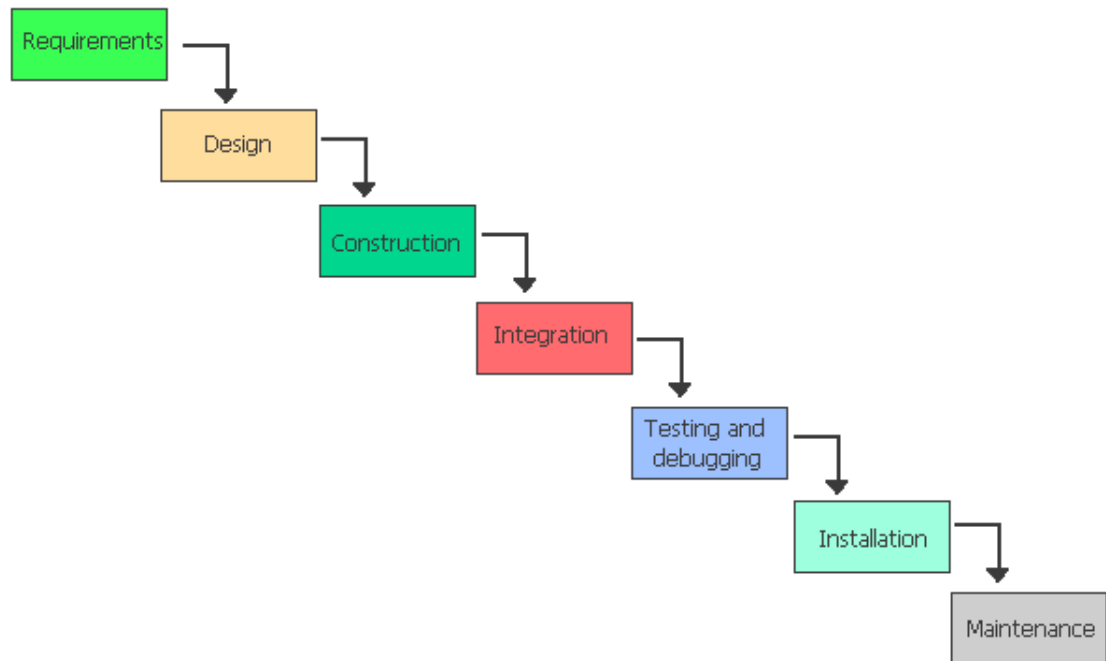


Figure 4.1: Waterfall Model

The origin of this model is attributed to Winston Royce in a paper he wrote on his views of how to manage large software systems, particularly in Spacecraft mission planning and post-flight analysis projects. Even if he recognizes there that he believes in the concept he also acknowledges that the implementation is risky and can lead to failure since the testing occurs at the end of the process and if there errors discovered at this stage the redesign is very costly. (Royce, 1970) It is ironic because Royce is credited with the origin of the model but the author did not use the term “Waterfall” and did not advocate its use either. He was describing the current status quo and promoting a more iterative model.

The biggest advantage of this model is that if the analysis and the requirements of a software project are clearly defined at the beginning of the software project there will be big gains in terms of the time and money during the implementation

phases. In terms of project management is also useful because it is easy to plan the beginning and the end of each phase and calculate the total cost of the project because all of the variables are considered before investing efforts into subsequent phases. Since the model is well structured and the phases are separated, a team member has a clear understanding of their role in each one of the phases and finally, the end product is claimed to have a high reliability (McConnell, 1996).

Unfortunately in real life software projects it is not very common to have a complete defined set of requirements at the beginning of the project and to have a rigid structure creates issues since the model does not reflect the ever changing reality of dynamic environments. In many software projects the client has a vague idea about what the system should do and once she sees some prototypes she can redefine exactly what she is looking for. Therefore one of the big disadvantages of this model is that the rigid implementation of the phases does not match with the changing nature of the majority of software projects

### 4.2.2 Agile Framework

The Agile Software Development framework is not a process *per se* but rather a framework that encompasses a different set of methodologies that use an iterative development, adapting to the changes in the software requirements, with self-organized and highly motivated teams that use the best engineering practices in order to deliver software quickly, of high quality and meeting the stakeholders expectations. In the Agile approach there is planning but it is less rigid and the specifications of the product evolve based on the feedback gather from the clients. Martin Fowler is one the founders of the movement and he describes in his website how a group of developers meet in 2001 to discuss some development lightweight methods and decided to encompass all of them under the “agile” term and how they wrote a manifesto that contains the main principles of the Agile Software De-

velopment (Fowler, 2006). Actually this manifesto is presented here:(Beck et al., 2006)

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools  
Working software over comprehensive documentation  
Customer collaboration over contract negotiation  
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Teams working in the Agile framework have their customers as the number one priority and they want to keep them happy by continuously releasing valuable software and since they use shorter scales in the development process they accept changes in the requirements (even at a later stage). Another typical feature of this framework is the communication between the team, since they prefer the face to face communication and to put together the business and technical team members. Finally they consider that the best products come from self-organizing teams and that the software should be the main parameter for considering the success of a project.

Some of the known development methodologies that can be classified as Agile are: Extreme Programming, Scrum, Agile Unified Process and Open Unified Process.

The main advantage of this methodology is that since it is customer oriented, this results in a highly satisfied end-user since they receive different iterations of the product with added functionality and since they can give feedback to the development team, most of their concerns are normally covered. The downside of these

methodologies is that there is always a risk of scope creep since they projects might lack the support documentation; the other problem is the expertise of the teams: in order to be successful an Agile team need to be composed by very good or senior developers, since the team are self-organizing and work in fast turn around time (Papadimoulis, 2007).

In conclusion the Agile Framework can be an excellent solution for the Software Development in an iterative fashion when the team is composed of highly motivated developers with a great degree of expertise who are able to deliver high quality software in successive iterations and who are able to quickly adapt the product to changes in the software specifications.

### 4.2.3 Rational Unified Process (RUP)

The Rational Unified Process is both a software engineering process and a process product. The idea behind this process is to provide a structured framework to assign responsibilities in a development team. The final goal is to produce software of a high standard meeting the the needs of the end user within the expected schedule and budget. (Kruchten, 2003)

The RUP is also a software product created by the company Rational Software (now this company is part of IBM), where Ivar Jacobson, Grady Booch and James Rumbaugh combined their knowledge and experience and incorporated popular object-oriented principles to define the best practices in the Software Development process. They identified the causes of the failures of projects using traditional methods like the waterfall model and they compiled these software best practices in the Rational Unified Process. The practices they identified as suitable for a wide range of projects and organizations include: (Kruchten, 2001):

- Develop software iteratively.
- Manage requirements.

- Use component-based architectures.
- Visually model software.
- Continuously verify software quality.
- Control changes to software.

Fowler believes that one the motivations driving RUP is to emulate the success of UML unifying the modeling languages into a single one, but the outcome was not a process *per se* but rather a framework composed by a large set of practices. The Rational Unified Process does not provide a unique process; instead it offers the developers a set of practices the team can chose from based on the requirements of a particular project. Nevertheless one of the problems he identifies with this framework is that it is extremely variable. He identified that some definitions present a model as rigid as the waterfall model whereas some others could be framed inside the Agile Development Manifesto (Fowler, 2005).

**Architectural Elements** The architectural principles of the Rational Unified Process are based on two dimensions that are illustrated in the figure 4.2. The horizontal dimension represents the lifecycle of the software process, including different cycles, phases, iterations and milestones. Since RUP uses an iterative approach, the phases are revisited in incremental iterations. The vertical dimension on the other hand represents the core process which are grouped by engineering activity so this axis contains tasks such as Business modelling, Requirements, Analysis and Design, Implementation, Testing, etc.

The project lifecycle in RUP has four phases allowing to the project to be seen from a higher perspective. These phases are iterative but each one of them has objectives and milestones to be accomplished. Those phases are(Kruchten, 2003):

- **Inception phase:** The main factors in this phase are the costing schedule, and budget. Here the Stakeholders must meet and agree on those variables by

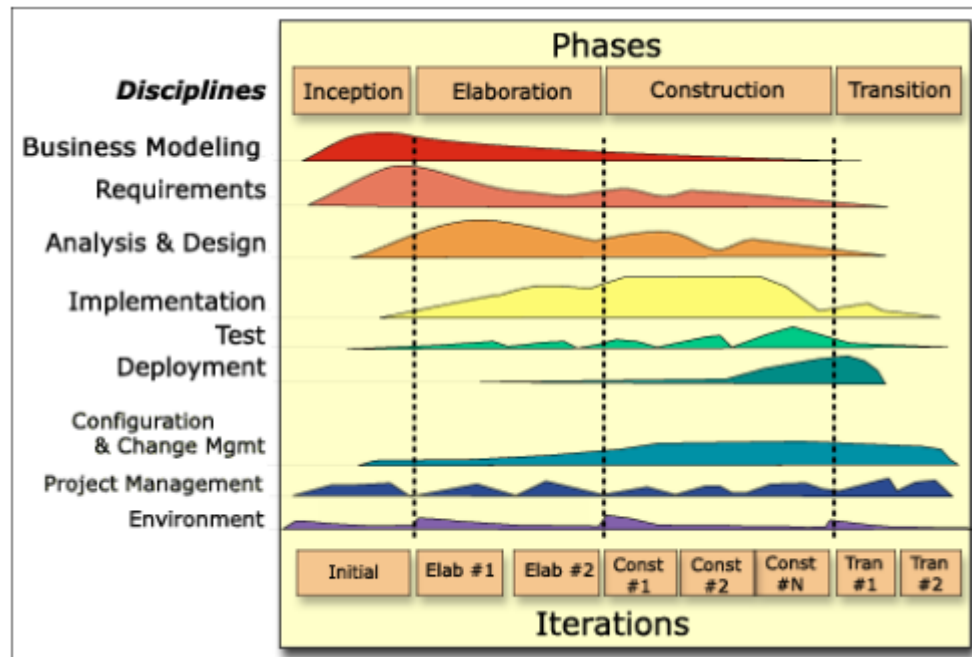


Figure 4.2: Two dimensions of the Rational Unified Process

evaluating factors such as business context, success factors and financial forecast. The project needs to pass what's called the lifecycle "objective" milestone. Otherwise it needs to be reevaluated.

- Elaboration phase:** The objective of this phase is to analyze the requirements and get an architectural model representing as much as possible the problem the project is going to solve. This includes use cases models, software architecture, business cases, development plan and prototypes. Therefore this phase produces the blueprint that will be used for the software development. The milestone of this phase is called lifecycle "Architecture".
- Construction phase:** The aim of this phase is to actually build the system. Here is where the main coding activities take place. At the end of this phase the team will produce the first version of the software. The milestone of this phase is named "Initial Operational Capability".
- Transition phase:** The objective of this phase is to make the transition be-

tween development and production, so the product is delivered to the end user. Some of the activities related to this phase are beta testing, training and validation against the architectural model created in the inception phase. If this phase is successful the “Product Release” milestone is achieved and the project ends.

The main advantage of the Rational Unified Process (RUP) is its iterative nature. RUP takes into account that the requirements of the product might change but it tries to manage requirements creep by keeping developing and integration of the elements in successive cycles. Another big advantage of this iterative methodology is that it is possible to make tactical changes to the product without having to redesign the whole software system, since the new features will be just incorporated in the next release. Finally the most significant benefit of RUP is its use-case-driven approach, meaning that use cases defined for the software can be used during the rest of the development process, so the design, testing, UI design will be tied to those use cases.

The only disadvantage identified with this methodology was discussed previously and it is that in certain cases the definitions of the RUP can be too variable and that in some implementations might be as rigid as the waterfall model (Fowler, 2005).

### **4.3 Development Process Selected**

After reviewing some of the most popular software development process and since one of the main objectives of this dissertation is to produce a Medical Social Bookmarking system using the principles of Web 2.0 where the principles of release early and release often are encouraged, one of the first criteria for choosing the software process is that it has to be iterative. Since the waterfall model is by definition



a sequential process where each one the steps have to be completed until the final release of the product it is not suitable for this project. Due to the dynamic nature of the web it is possible that during the development phase of the product a new technology might appear that could be interesting to add the to the final product but the waterfall model imposed restrictions on new features once the project is in the implementation phase.

The selected process has to be then an iterative one, therefore the Agile Framework and Rational Unified Process in principle could be considered as good candidates for the development of this social bookmarking service. Some of the elements of the agile manifesto such as considering the working software as the main parameter of success, focus on the customer and release the software in smaller timeframes are appealing, nevertheless some of the elements like constant meetings between the teams could not be used here because there is only one developer in this project. On the other hand some of the elements of the Rational Unified Process such as developing the software iteratively, to use component-based architectures and to visually model software using the UML language is also suitable for the project. One interesting feature of the Rational Unified Project is its use-case-driven approach, so for this project it could be useful to define some clear use cases and then adapt the development based on them. One of the disadvantages of RUP is that it is also a product and I will not have access to it. Since this is a small size development project with only one active developer, a striped down version of the Rational Unified Process with a very agile implementation seems more appropriate where the use cases and analysis requirements can be modelled using the UML notation, but where the development and integration will be done in small iterations, implementing different use cases in each iteration and releasing soon and releasing often.

## 4.4 Conclusion

In this chapter the different methodologies and structures used for developing or maintaining a software project were reviewed. One conclusion from this reviewed is that most of the methodologies can be splitted in two big groups the sequential and the iterative ones. The waterfall model was reviewed as the most significant representative of the sequential group and it was decided that the rigorous steps that have to be followed convert it in a challenging option for the dynamic Web Applications world. The iterative methods that were discussed are the Agile Framework and the Rational Unified Process (RUP). Those two methodologies encourage software development in an iterative fashion and accepts changes in the user requirements, but RUP also encourage the use of the visual modelling of the software and it is use case driven. In this project I will not have access to the RUP product but it was decided that for the project of this dissertation some of the elements, particularly the ones of “Visualising and Modelling” the software can be used so, a scaled down version of the Rational Unified Process in an agile development framework was chosen.

# Chapter 5

## Analysis and Design

*‘UML is not dessert topping and floor wax (Grady Booch).’*

### 5.1 Introduction

In the previous chapter the Waterfall, Agile Framework and the Rational Unified Process were discussed and assessment about the advantages and disadvantages of each one them was conducted to determine which one of the alternatives was the most suitable one for this Social Medical Bookmarking system. In this chapter the project cycle will start with a high level analysis of this business problem to solve, then an analysis of the most suitable architecture for this project will be discussed, an analysis of the functional requirements will be performed and the system will be modeled using standard UML notation. An Entity Relationship diagram of the datamodel will be also presented at the end of this chapter.

### 5.2 High Level Analysis

The objective of this Software Development Project is to create a social bookmarking service for the medical community that will run in a Web environment as a website.

This website will allow its user to store their favorites websites as bookmarks with a title and a set of tags. This website will be accessible mainly by web browsers installed in Windows, MacOSx or Linux based personal computers, but the site will be also accessible by other devices such as PDAs, smartphones and Blackberries. The non registered visitors will be able to browse the system click in the different tags and see the bookmarks registered by other users and associated with that particular tag, they can also search the bookmark repository by using keywords. The registered users will have the same functionality described before but additionally they will be able to add new bookmarks into the repository. Here is a sample description of this functionality from a typical medical user:

*“I’m a Medical Doctor browsing the web looking for news and information about Swine Flu. Using a search engine I found an official site from the US government, PandemicFlu.gov that provides information on the pandemic influenza and the avian influenza so I want to keep it for later reference. I go to my medical social bookmarking site, I sign in and I click in a new a bookmark and I store this website with the title Pandemic Flu US Goverment, and I tag the site with the keywords reference, govermental, influenza, flu and swine”*

So this registered medical user will have a personal account with a list of all the bookmarks they have added, but they can also browse some other people’s bookmarks. Based on the concepts of Hotho et al the set of bookmarks of each one of registered users will be consider a “Personomy” and the combination of all this personomies will generate the “Folksonomy” of the system (Hotho et al., 2006a). Finally there will be also an Administrator User who will be able to perform the same actions explained before but additionally she will be able to delete entries from the database (for example spam entries) or delete users from the system (spam users).

## 5.3 Architecture

In this section the different architecture models used in the Web Applications will be discussed. The concept of Model View Controller architecture will be explained and the different Web Application frameworks will be covered.

### 5.3.1 Model-View-Controller Architecture

The model view controller was described by G. E. Krasner and S. T. Pope as an application architecture where the application domain is separated from the interface and from the user interactions modifying the underlying application domain. In their architecture the “model” represents the information or the domain structure and in most of the cases it is related to the “data” layer encapsulated by this model. The “view” normally displays the application’s state, so it provides a graphical user interface where the user can interact with the application and modify the data defined by the model and finally the response to the user interaction is carried out by the “controller” that handles the relationship between the view and the model. They observed in their previous experience that separating the model of the application domain from the model that was shown to the user and the way she could interact with it made sense since it was offering a high degree of modularity. For example the developers could focus on just one component without having to understand other areas. The relationship between the different components is depicted in the figure 5.1 (Krasner & Pope, 1988). In conclusion this pattern emphasizes the separation of the application model, the Graphical User Interface (also called the view) and the control logic managing all the different parts of the application (see Figure 5.1). Even if the concept itself has been around for almost 30 years, recently with all the developments of in the web application domain, the concept has been implemented in several web application frameworks.

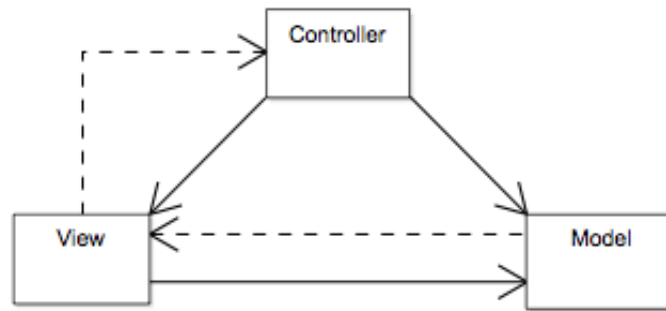


Figure 5.1: Model View Controller

### 5.3.2 Web Application Frameworks

In Chapter 2 which discussed Web Applications and Web 2.0 Applications, how the architecture of the web can be summarised in the transfer of HyperText Markup Language (HTML) documents from one server to a client using the Hyper Text Transfer Protocol (HTTP). Actually at the start the web was planned as a media to transfer static marked up documents so web developers needed to hardcode each one of the pages and upload them in the webserver. The Common Gateway Interface (CGI) was the first attempt to give the developers the ability to create pages dynamically using programming languages like Perl or C/C++ (NCSA University of Illinois, 2001), but the capabilities were rather limited and these models lead to the creation of not very secure applications. Then some real web programming began to emerge when systems developed in industry supported languages like Java and C, using different layers of abstraction to accomplish the tasks of bridging the logic of the business with the HTTP protocol transmission. J2EE was one of the first frameworks trying to put everything together, using the concept of servlets. Even if those frameworks were very secure and made use of good programming practices, the activities involved in creating a web application on them were complex with many steps and configuration files (Daly, 2007).

At the same time appeared some lightweight script languages created specifically for generating those dynamic contents on the fly such as Coldfusion, followed

by ASP (Active Server Pages) and PHP (Hypertext Preprocessor). The last one in particular became very popular because of its open source nature and its easy integration with the popular Apache Web Server (nexen.net, 2008).

Even if those languages were a big step forward in creating dynamic websites, soon many developers realised they were rewriting over and over again the same functions to authenticate the users, establish database connections, generating the output HTML code, etc.

That gave the idea to many developers at the same time to create “Packages” that could manage all these repetitive tasks common to most the web applications, so the developer could focus on the logic of the problem. Therefore the concept of Web Framework was born as a collection of software packages or modules created to help the web developers to write Web applications or Web services without having to deal with low-level issues as protocols, authentication, database connection, etc. Many of those Web Frameworks adopted the Model View Controller Architecture principles. Some of the tasks managed by the Web Frameworks are:

- Interpreting HTTP requests,
- Authentication,
- Database connections,
- URL mapping and rewriting,
- Template system.

There are a broad array of web application frameworks for most of the programming languages such as PHP, smalltalk, ASP, Cold fusion, Perl, Java, Ruby and Python. Most of this frameworks are Open Source, but some are proprietary. The Mindtree blog has a good compilation of the most popular Web Application Frameworks (Mind Tree Blog, 2008):

- **PHP:** Zend Framework, CakePHP, Symfony, CodeIgniter, Akelos and Prado
- **Java:** Google Web Toolkit, Spring Framework, Apache Cocoon, JSF - JavaServer Faces, Aranea, AppFuse and Struts.
- **Ruby:** Ruby on Rails, Nitro, Camping and Ramaze.
- **C Sharp:** The .NET Framework, MonoRail and AForge.NET
- **Python:** Django, Pylons, TurboGears and Gluon

### 5.3.3 Django

#### Overview

Django is an open source high-level Python Web Application Framework that encourages “rapid development and clean, pragmatic design”. This application framework was developed by web developers of a fast-paced newsroom environment (World Company in Kansas) who decided to switch from PHP to Python and create a set of packages to make the routine Web-development tasks quick and painless. The framework was released under the BSD license but in June 17 2008, Jacob Kaplan-Moss one of the original developers of Django, announced the creation of the Django Software Foundation to support the development, promote the use and protect the intellectual property of the Django Framework ([djangoproject.com](http://djangoproject.com), 2008).

The history of Django Web Application framework is explained in the Django Book and basically it describes a common problem faced by many professional web developers; How to create and maintain many websites at the same time with tight schedules without having to rewrite over and over the same code: *“Django grew organically from real-world applications written by a Web development team in Lawrence, Kansas. It was born in the fall of 2003, when the Web programmers at the*



*Lawrence Journal-World newspaper, Adrian Holovaty and Simon Willison, began using Python to build applications. The World Online team, responsible for the production and maintenance of several local news sites, thrived in a development environment dictated by journalism deadlines. For the sites including LJWorld.com, Lawrence.com, and KUsports.com journalists (and management) demanded that features be added and entire applications be built on an intensely fast schedule, often with only days or hours notice. Thus, Adrian and Simon developed a time-saving Web development framework out of necessity it was the only way they could build maintainable applications under the extreme deadlines.”(Holovaty & Kaplan-Moss, 2007)*

Some of the main highlights of the Django Framework are (djangoproject.com, 2008):

- **Object-relational mapper:** It allows the developer to define the data models as Python Classes and the framework connect to the database, issue the SQL queries and create or update the tables.
- **Automatic admin interface:** Django creates administration interfaces out of the box for adding and updating content.
- **Elegant URL design:** In certain languages like PHP and ASP the URL structure is defined by the language, but Django gives the developer the flexibility to define the URL structure she decides.
- **Template system:** The Django framework comes with a template system and an intuitive template language that allows the author to separate the logic from the presentation

### Architecture & Backend Integration

The core of the framework is composed by four components: An Object-relational mapper that acts as intermediary between the Python created models and the re-

lational databases, and URL dispatcher that supports regular expressions, a view system that processes the requests and execute the business logic and finally a template system. Additionally the frameworks contains a testing development server and contains some useful model to create a validate forms. Finally the framework also offers a set of middle-ware classes that runs a every time that Django handles a request (Holovaty & Kaplan-Moss, 2007).

Django can run with different server environments. For example the popular Apache Server is supported using `mod_python` or `mod_wsgi`. Also for the Object-relational mapper python supports PostgreSQL, MySQL, SQLite and Oracle.

A diagram of the steps of the typical Django application are illustrated in the Figure 5.2. In summary they are:

1. The client browser asks for a URL and the web server passes the request to the URL dispatcher of the Django framework.
2. The URL dispatcher passes the request to a view matching the URL structure. If there is a cached version it will be returned to the web server and served to the browser, otherwise the view will process the request.
3. If the views needs to read or store information into the back-end, it will communicate with the model that will map the Python models with the relational database system. The result of the query is sent back to the view.
4. When the view finishes to process the request, passes the information to the template system, with the name of the HTML template to use. The template system selects the html template and binds the information from the view into the template.
5. The resulting document is passed to the web server and the server sends the document to the user.

One final note related to the server and backend integration is that it is possible to use Django very easily with the Google App Engine application hosting platform.

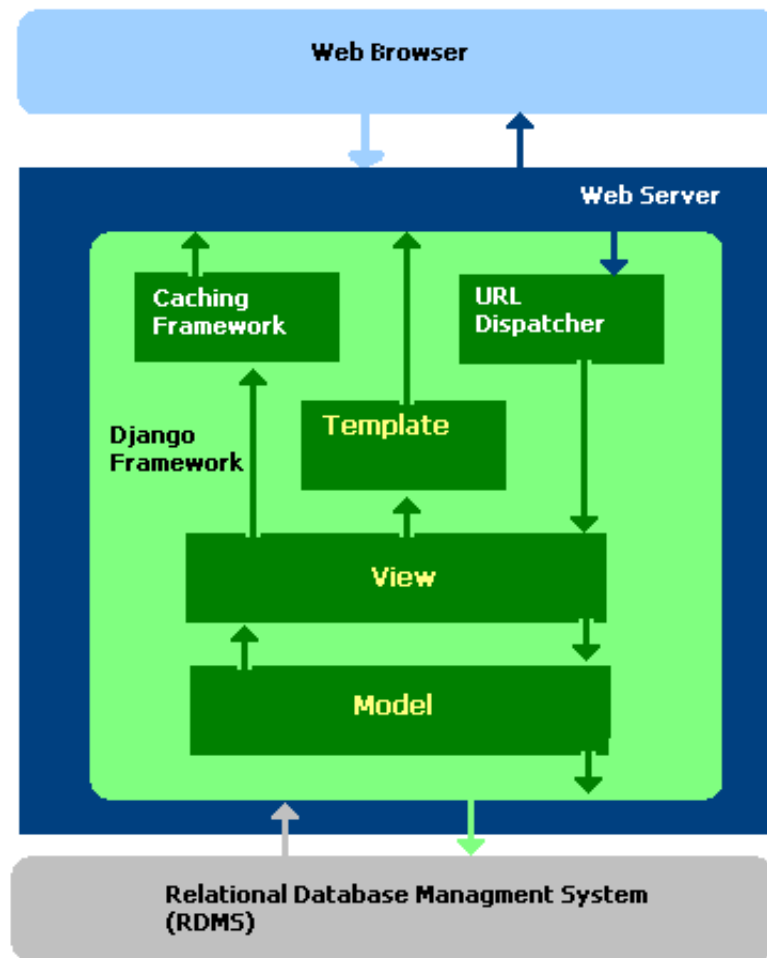


Figure 5.2: Django Framework Functionality.

Both Google App Engine and Django are designed to support the WSGI so it is possible to use most of the functionality of Django out of the box in the Google App Engine. It is only necessary to adapt the data models to make them compatible with datastore. Actually the Google App Engine includes the Django version 0.96. as one of the standard frameworks.

### 5.3.4 Google App Engine

The Google App Engine is a service that allows a web developer to create and host Web applications written in the Python programming language.<sup>1</sup>

The engine not only supports the Python standard library, but also Application Programming Interfaces (APIs) for the datastore, Google Accounts, URL fetch, image manipulation, and email services. The Google App Engine also offers a Web-based Administration Console where the developers can monitor and manage their running Web applications. Currently, the Google App Engine is free to use with up to 500MB of storage and about 5 million page views per month (Google.com, 2008c).

Any kind of web application written in Python can run over the App Engine platform, but the system is designed to host particularly applications user by many user at the same time. The system is designed to scale since the application will be running in the on Google's infrastructure so if the application becomes popular and receives a significant amount traffic or it is used by many users, the hosting platform can handle it. As more people use the application, App Engine allocates more resources and all this process is invisible to the user. Some of the features of the platform that were discussed once the service was launched are (Google.com, 2008d):

- Persistent storage (powered by Bigtable and GFS with queries, sorting, and transactions)
- Automatic scaling and load balancing
- Google APIs for authenticating users and sending email
- Fully featured local development environment

---

<sup>1</sup>Note: At the moment of finishing this dissertation Google announced it was supporting the Java programming language for the App Engine platform (<http://googleappengine.blogspot.com/2009/04/seriously-this-time-new-language-on-app.html>).

One unique characteristic of the Google App Engine compared to other hosting platforms is the backend used for handling the data since they only permit to store and retrieve it from a BigTable non-relational database. BigTable is a compressed, high performance, and proprietary database system that runs over on Google File System (GFS)(Chang et al., 2008). Finally the App Engine platform offers a Software Development Kit (SDK) available for Windows, MacOSx and Linux, including a development web server to testing the application locally.

As was discussed in the previous section Google App Engine supports natively the version 0.96 of Django, so based on the architecture of the Google App Engine and expanding the diagram used there, the Django framework inside the App Engine is illustrated in the Figure 5.3

### 5.3.5 Architecture in Gray's 2.0

Henry Gray was a famous English anatomist, known for his methodical approach to the knowledge of the structure and workings of the human body. He is the author of the classical book of Anatomy. Since the objective of this project is to create a Social Bookmarking system to tag medical resources on the World Wide Web, “Gray” and “2.0” would be good tags to bookmark the actual service, so “Grays 2.0” will be the code name of the web application.

Grays 2.0 is a Medical Social Bookmarking application that will run in the World-Wide Web, it seems reasonable to separate the presentation part (HTML pages, CSS styles) from the logic of the application (HTTP request processing, database access, etc). Therefore a web development framework incorporating the elements of the model view controller seems the most suitable option.

Python is known for having a extremely clean syntax and have a strong object orientation. This language has been used in agile projects, so a Python-based framework is probably the best alternative for this project. Django is one of the most

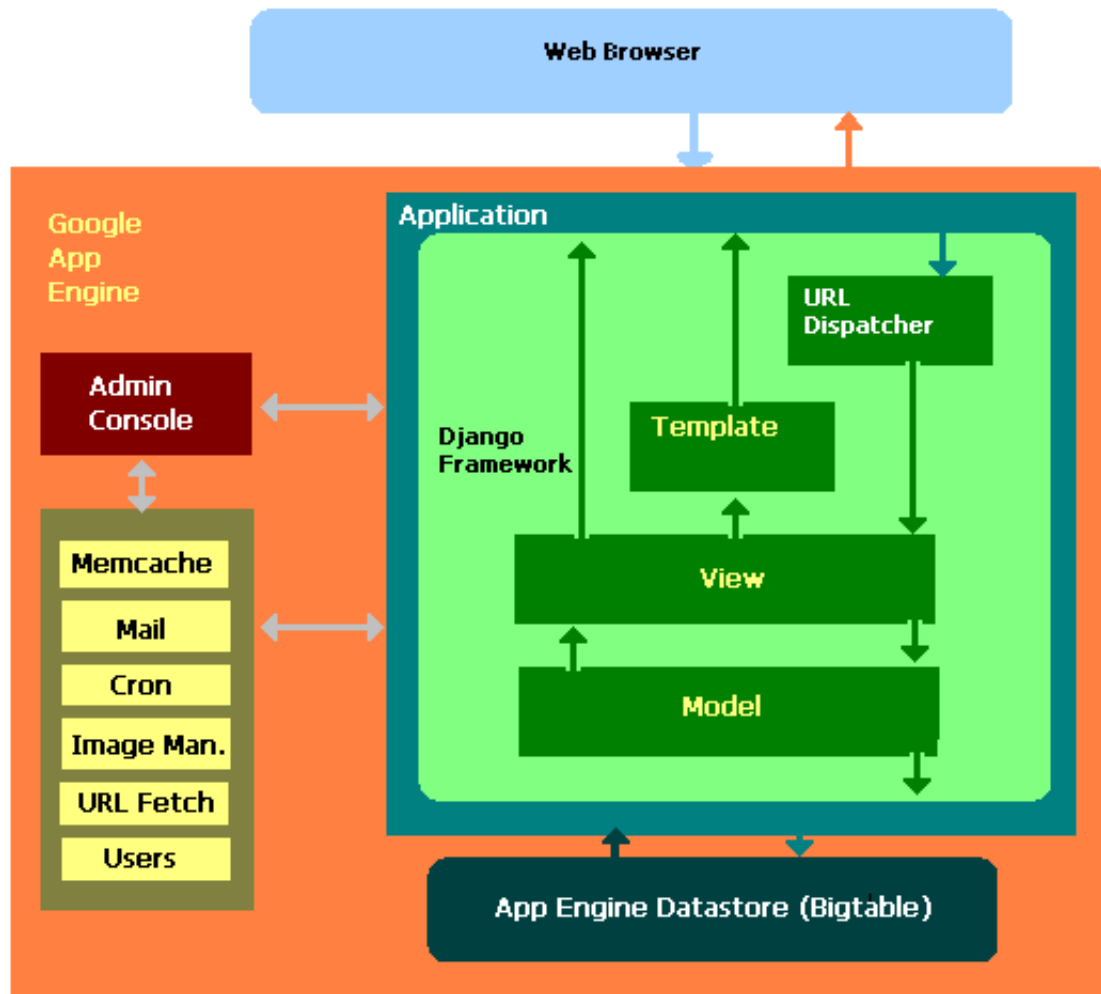


Figure 5.3: Django Framework on the Google App Engine.

popular Python Web Application Frameworks with one with a broad community of users. Django not only separates the logic from the presentation, but also offers an elegant implementation of the URL structure that can be useful for managing the multiple user and tag pages that will be generated once the systems grows.

Finally the Google App Engine is considered as a good hosting alternative for this project because it is designed to scale, so the system could be able to handle a growing number of users and page views. Nevertheless the App Engine offers some quotas for free and also gives to possibility to have a sub-domain under the appspot.com domain without any cost. Additionally the software development kit

is available to download easily and supports Django 0.96 out of the box. Therefore the alternative of developing Grays 2.0 in Django over the Google App Engine seems to be the most effective solution.

Based on the architecture discussed in the Django section, the design of this application will be based on the methods that will be implemented on the “views” module, where the logic of the business resides in a Django project. The Figure 5.4 shows the different elements that will be constructed during the development time.

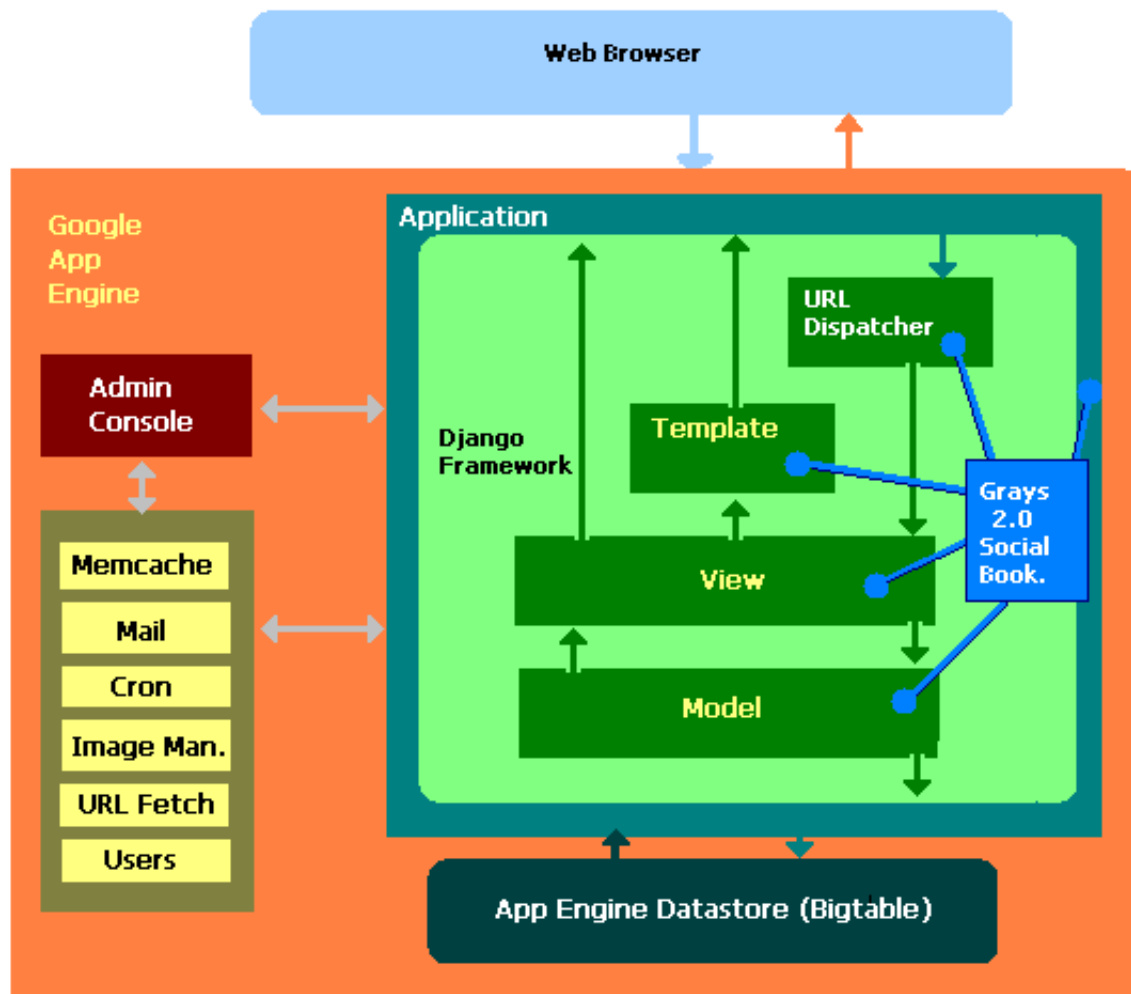


Figure 5.4: Grays 2.0 on the Django Framework over the Google App Engine.

## 5.4 UML Diagrams

### 5.4.1 Use Cases

Use Case diagrams identify the functionality that will be offered by the system, it describes the users who will interact with the system, and the association between the users and the system. Use Cases are normally used in the Analysis phase of software development cycle to create the high-level requirements of the application.

As was discussed in the previous section the system will contain a big repository of bookmarks, each one them composed by an URL, a title and a set of tags associate with a bookmark. Based on the previous analysis we can identify in the Social Bookmarking system three different kind of actors. Most of the use cases are common to most the actors but each one of them have an specific set of cases:

- **Visitor:** A visitor is considered a guest actor in the system that either has not been registered yet or has not logged in the application. A visitor can browse the bookmarks by tags, browse the bookmarks added by an registered user and search the bookmarks by keyword.
- **Medicals User:** A medical user is considered a medical actor that has logged in and therefore has an account in the social bookmarking system. The main Use Case added to this type of user is the ability to add a new bookmark and tag the resource with the favorite keywords. The user actor can also browse bookmarks by tags, browse the bookmarks added by other registered users and can search the bookmarks of the repository by keyword.
- **Administrator:** The administrator actor has also an account in the system and therefore needs to be logged in in order to perform the actions associated with the role and has the same use cases of the Medical User actor but additionally she can remove bookmarks, tags or users from the system (mainly because of



spam issues).

Now each one of the previous use cases will be analysed separately and the UML Use Case diagrams of each one of them will be included:

**Create a new user account**

|   |   |
|---|---|
| <b>Use Case Number:</b> U1  | <b>Use Case Name:</b> Create a new user account and manage user accounts. |
| <b>Goal:</b> To create a new account for a new customer and manage the accounts.  |   |
| <b>Brief Description:</b> A new user has to register for use any service in the Grays 2.0 system. The user fills a form asking for a new account and then the User Administrator creates the new account. The administrator can edit and delete the accounts.   |   |
| <b>Actors:</b> Administrator, Medical User  |   |
| <b>Frequency:</b> High. Each time a new user browses the website a new account has to be created. The administrator will manage the accounts frequently   |   |
| <b>Scalability:</b> If the site is successful it is expected that the user mass will grow and there will be new registrations.  |   |
| <b>Criticability:</b> Very. All the new users have to register so without this case it is not possible to add new users.  |   |
| <b>Primary Path:</b> <ol style="list-style-type: none"> <li>1. A new user browses the Grays 2.0 homepage and before using any service she is asked to sign in.</li> <li>2. If she does not have a username and password she is asked to register.</li> <li>3. Once she has filled all the required fields this information is passed to the system.</li> <li>4. If the validation was successful the system creates a new account for the user with the username and password.</li> <li>5. The user is authenticated with this information and can continue.</li> </ol> |   |
| <b>Use cases Related to Primary Path:</b> None.   |   |
| <b>Alternative Path:</b> 2.1 The information provided by the user is incomplete. No user account is created. She is invited to repeat the signup process again.   |   |
| <b>Use cases related to alternative:</b> None.  |   |
| <b>Exceptions:</b> None.  |   |
| <b>Use cases related to exceptions:</b> None.   |   |

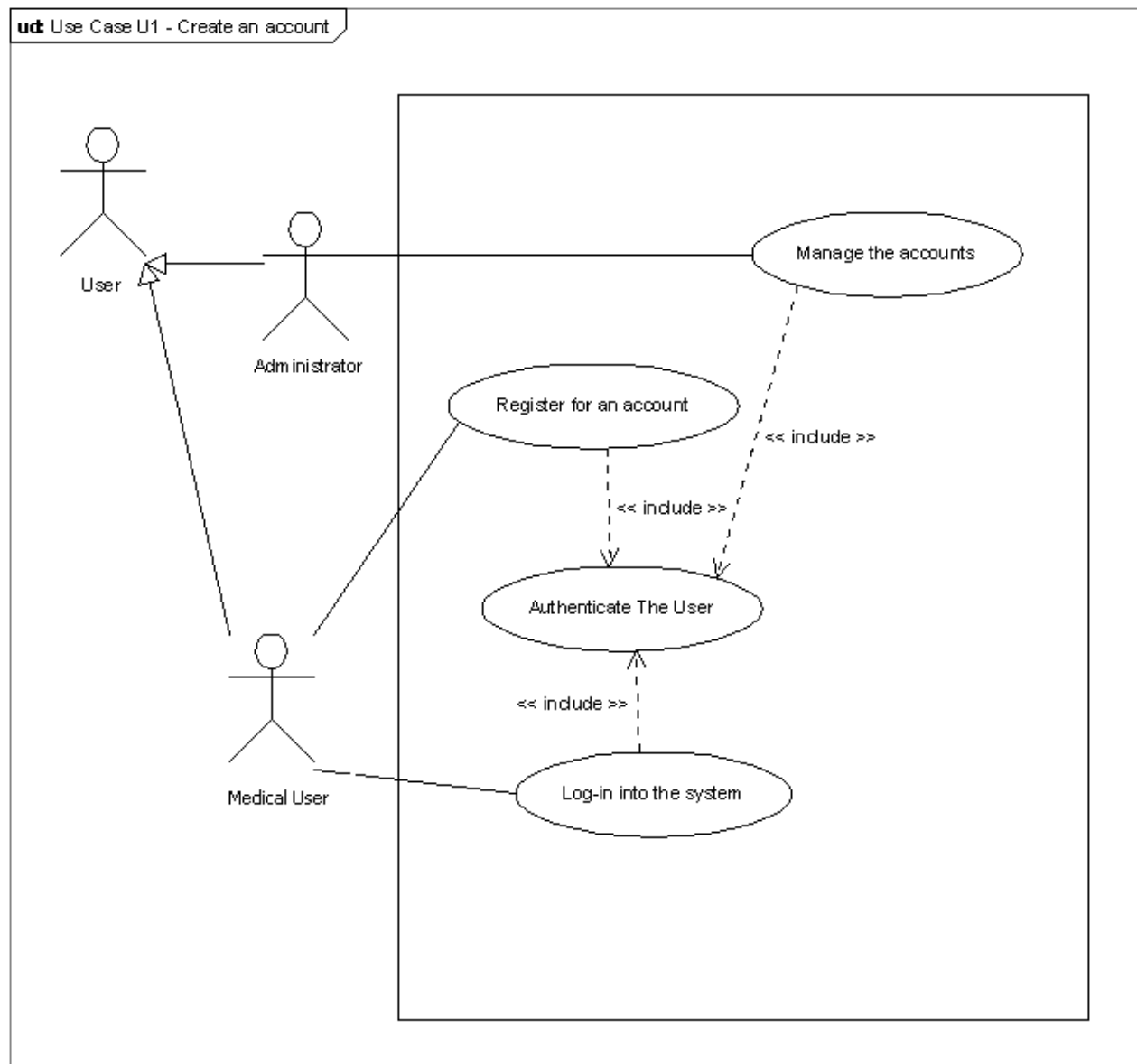


Figure 5.5: Use Case Number1: Create a new account.

**Add a new bookmark**

|  |   |
|--|---|
| <b>Use Case Number:</b> U2   | <b>Use Case Name:</b> Add a new bookmark to the system. |
| <b>Goal:</b> To allow the registered user to publish new bookmark into the Grays 2.0   |   |
| <b>Brief Description:</b> Since the contents of Grays 2.0 are user-generated this main use cases allows the registered user to login into the system and publish a new URL, adding a title and associating the bookmark with the desired tags.   |   |
| <b>Actors:</b> Medical User  |   |
| <b>Frequency:</b> Medium - High. Even if not all the visitors will convert into registered users, ideally most of the users of Grays 2.0 will also be collaborators. Therefore it is expected that this Use Case will happen frequently  |   |
| <b>Scalability:</b> If the number of user begin to grow as expected there will be more users adding bookmarks. Nevertheless at the beginning the peek user number is expected to be approximately 15 users entering bookmarks at the same time.  |   |
| <b>Criticability:</b> Medium. The system is still usable even the adding bookmark section is down, but since the website is mainly about bookmarking websites, if this case fails it will compromise an important functionality.   |   |
| <b>Primary Path:</b> <ol style="list-style-type: none"> <li>1. A registered user browse the Grays 2.0 homepage and she is asked to signin.</li> <li>2. Once she is signed the option to add a new bookmark is offered.</li> <li>3. The user selects a title for the bookmark, enters the URL and associates the tags wanted.</li> <li>4. The bookmark is stored into the bookmark repository and the tags are associated with the bookmark.</li> <li>5. The user recieves a message confirming the the bookmark has been saved.</li> </ol> |   |
| <b>Use cases Related to Primary Path:</b> None.  |   |
| <b>Alternative Path:</b> 2.1 If the signin information is not valid, the user will be invited to the sign-in process again.<br><br>2.2 If the URL entered into the system is not a valid URL, the user will be informed and invited to enter a valid URL.  |   |
| <b>Use cases related to alternative:</b> U1.   |   |
| <b>Exceptions:</b> None.   |   |
| <b>Use cases related to exceptions:</b> None.  |   |

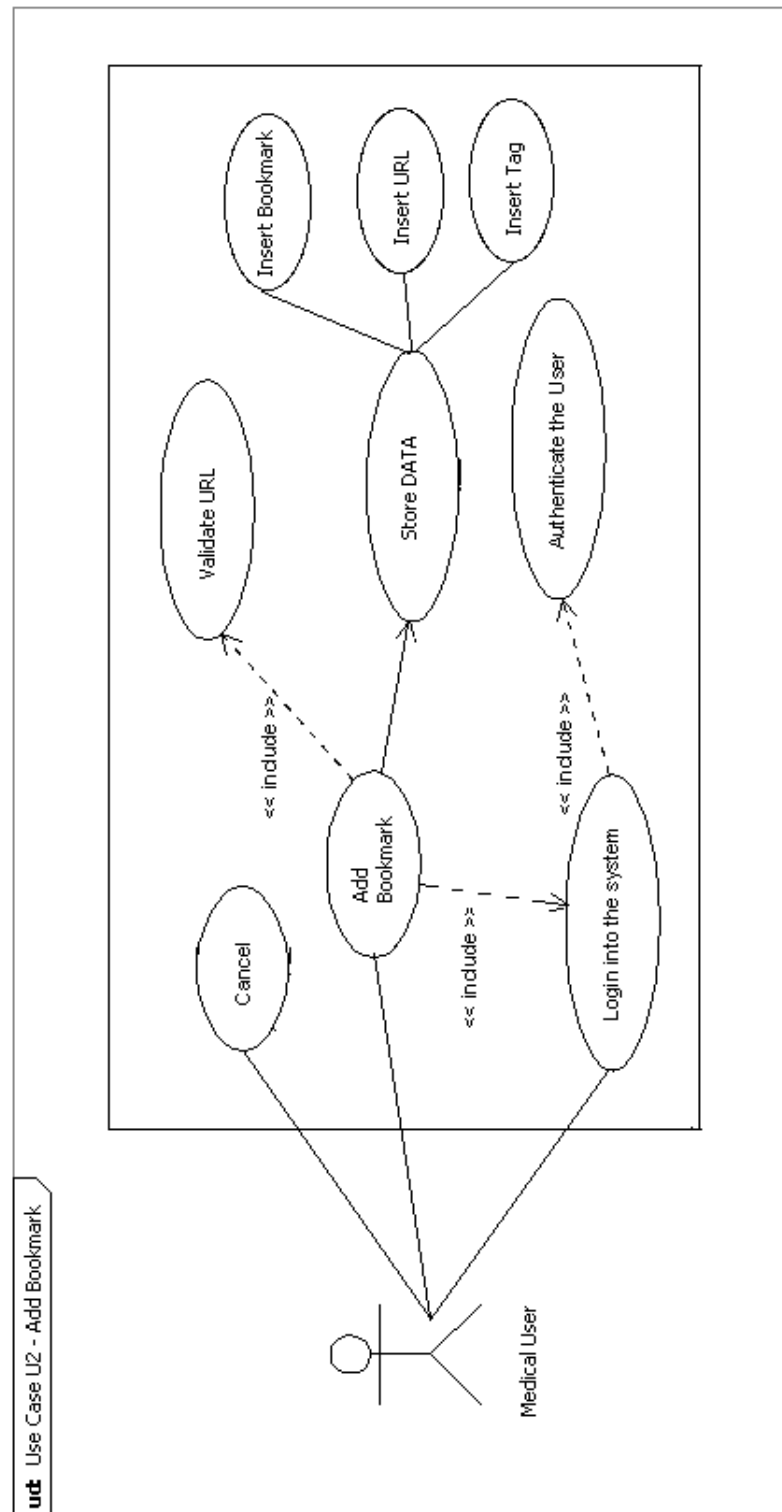


Figure 5.6: Use Case Number2: Add a new bookmark.

**Search for bookmarks**

|   |  |
|---|--|
| <b>Use Case Number:</b> U3  | <b>Use Case Name:</b> Search for a bookmark. |
| <b>Goal:</b> To allow the registered users or visitors to search for bookmarks.   |  |
| <b>Brief Description:</b> The user can look for bookmarks related to a query composed of keywords. The system will look in the bookmark repository and return matches for that query.   |  |
| <b>Actors:</b> Visitor, Medical User  |  |
| <b>Frequency:</b> High. It is expected the user will look for information several times on a daily basis, since the web users are very familiar with the concept of search engine.  |  |
| <b>Scalability:</b> At the beginning the peak user number is expected to be around 100 users looking for information at the same time.  |  |
| <b>Criticability:</b> Medium. It is not the only mechanism for finding information in the system, but it the functionality might be useful to the majority of users.  |  |
| <b>Primary Path:</b> <ol style="list-style-type: none"> <li>1. A registered user or a visitor browse any page in the Grays 2.0 system.</li> <li>2. A search box is presented where the user can enter the query.</li> <li>3. The user enters the query and submit the form.</li> <li>4. The system returns a list with one or more bookmarks matching the query.</li> <li>5. The user can browse the bookmarks or enter a new query.</li> </ol> |  |
| <b>Use cases Related to Primary Path:</b> None.   |  |
| <b>Alternative Path:</b> 2.1 If the query does not match any result the system will return a message informing the user and the search option will be offered again.  |  |
| <b>Use cases related to alternative:</b> U3.  |  |
| <b>Exceptions:</b> None.  |  |
| <b>Use cases related to exceptions:</b> None.   |  |

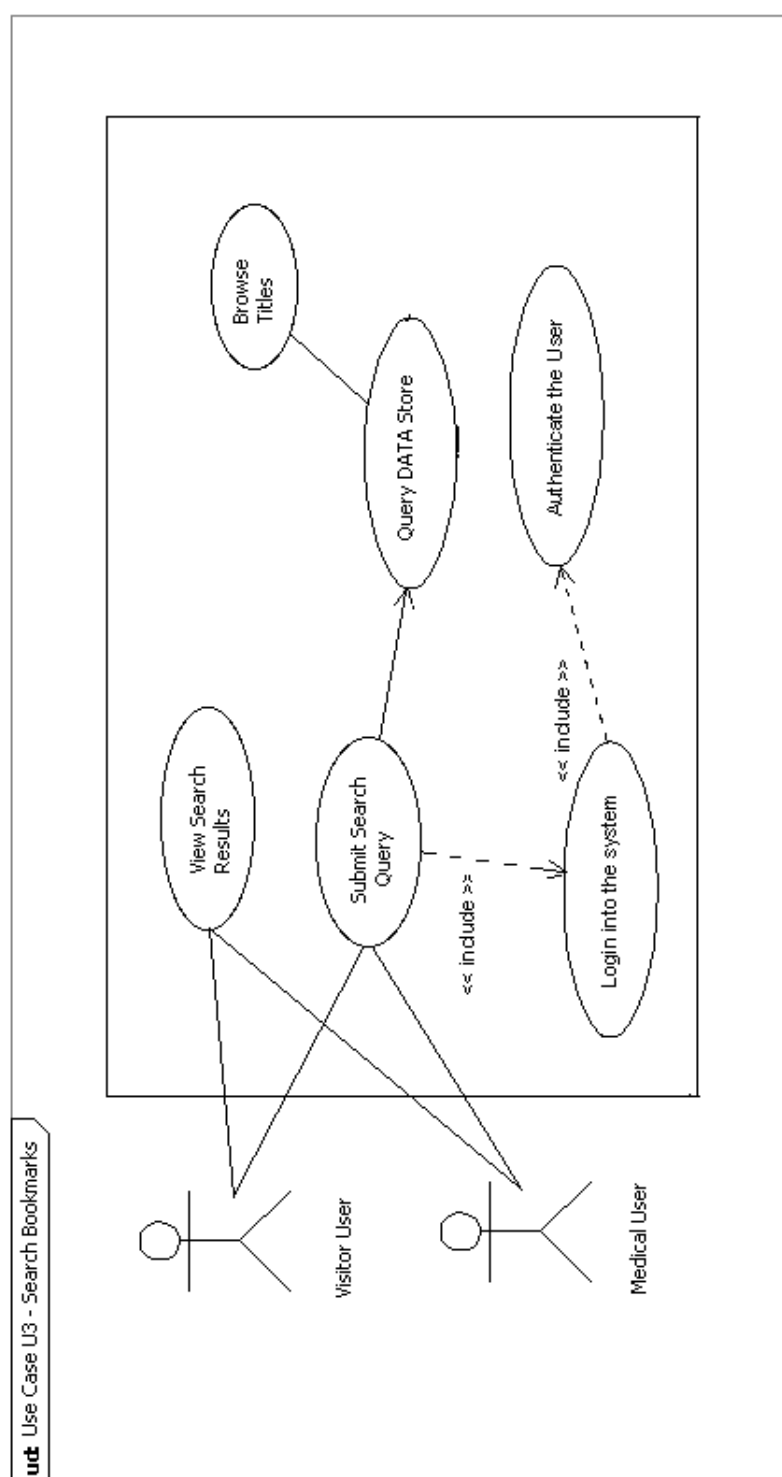


Figure 5.7: Use Case Number3: Search for bookmarks.

**Browse User Bookmarks or Tags**

|   |  |
|---|--|
| <b>Use Case Number:</b> U4  | <b>Use Case Name:</b> Browse User Bookmarks or Tags. |
| <b>Goal:</b> To allow the the registered users or the visitor to broswre the bookmarks by name or by tag.   |  |
| <b>Brief Description:</b> The user registered user can browse the system and get the book- marks she stored before. Both the registered users and the visitor can browse the book- marks created by other registered users or alternative they can browse the tag cloud an discover the most popular tags.  |  |
| <b>Actors:</b> Visitor, Medical User  |  |
| <b>Frequency:</b> High. This is one the main features of the Social Bookmarking system, to browse the bookmarks by user or tag so it is expected this features will be used very frequently.  |  |
| <b>Scalability:</b> At the beginning the peek user number is expected to be around 100 - 150 users browsing the bookmarks at the same time.   |  |
| <b>Criticability:</b> High. This is one of the main feature of the system. If the registered user can not get back the bookmarks she has stored before that is a major issue. Also it will be common for the visitors to browse the tag cloud frequently so the criticability of this use case is high.   |  |
| <b>Primary Path:</b> <ol style="list-style-type: none"> <li>1. A registered user or a visitor browse any page in the Grays 2.0 system.</li> <li>2. If the registered user is logged in there will a link to browse her bookmarks.</li> <li>3. Alternative the registered user or the visitor can browse some other users bookmarks by name.</li> <li>4. Finally they can visit the tag cloud and find the most popular tags.</li> </ol> |  |
| <b>Use cases Related to Primary Path:</b> U2.   |  |
| <b>Alternative Path:</b> 2.1 If the selected user has not saved any bookmarks yet the system will return an error message. If there are no bookmarks in the system both the user and tag cloud options will return this error message.  |  |
| <b>Use cases related to alternative:</b> None.  |  |
| <b>Exceptions:</b> None.  |  |
| <b>Use cases related to exceptions:</b> None.   |  |



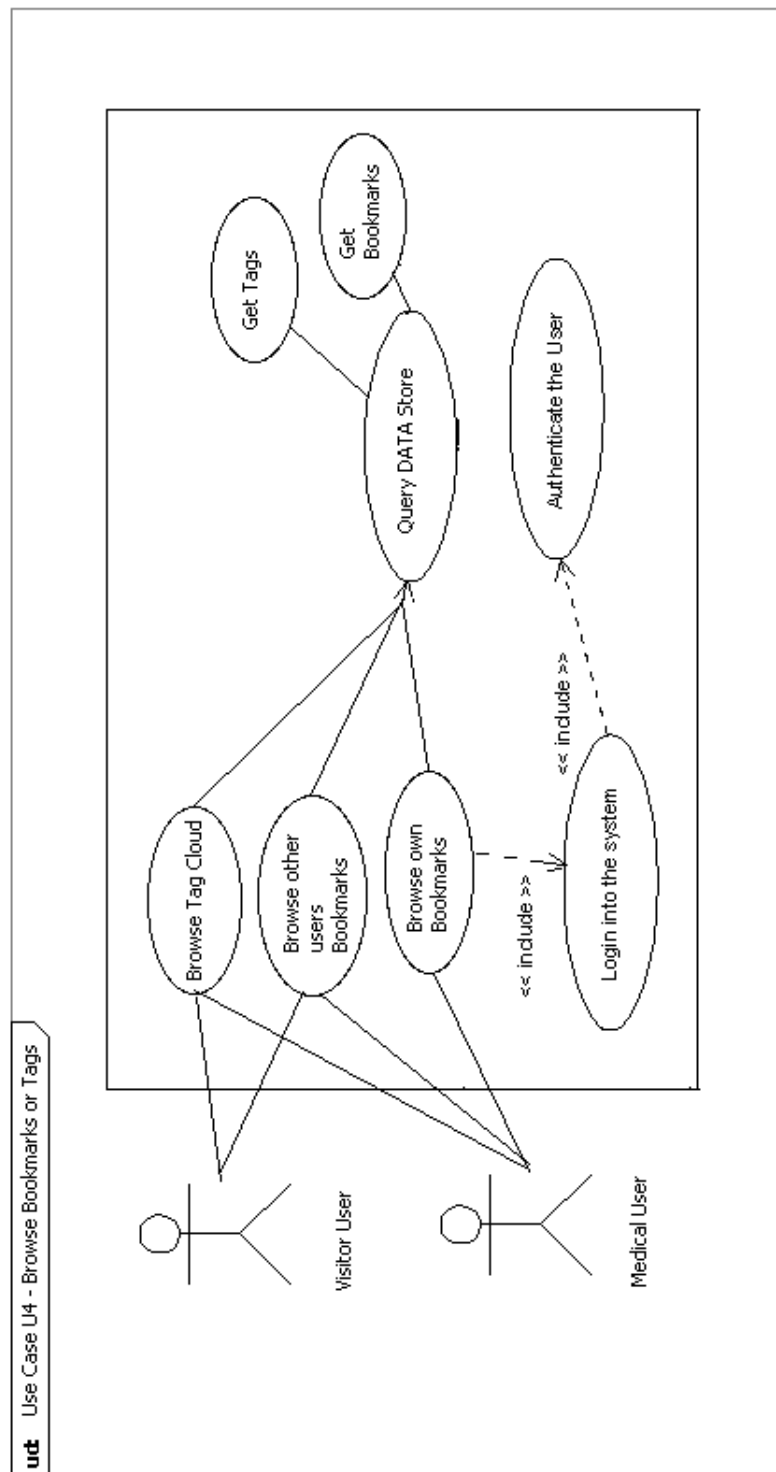


Figure 5.8: Use Case Number4: Browse bookmarks by user or tag cloud.

**Delete Contents**

|  |  |
|--|--|
| <b>Use Case Number:</b> U5   | <b>Use Case Name:</b> Delete Contents. |
| <b>Goal:</b> To give the administrator the tools to delete bookmarks, tags and links.  |  |
| <b>Brief Description:</b> As was discussed in a previous chapter on the risks of this system is to have spam users adding spam entries to the bookmark repository. The objective of this use case is to give the administrator the ability to erase entries from the data repository.  |  |
| <b>Actors:</b> Administrator   |  |
| <b>Frequency:</b> Low. Since the user will have to register before adding a bookmark it is expected that at the beginning this case will be used sparingly.  |  |
| <b>Scalability:</b> At the beginning the administrator will use this case only 1 or 2 times a week.  |  |
| <b>Criticability:</b> High. Even if the frequency of this case is low, it has a high because the spam entries should be deleted at once.   |  |
| <b>Primary Path:</b> <ol style="list-style-type: none"> <li>1. A administrator browse the Grays 2.0 system and log-in.</li> <li>2. She will see a whole list of users with the associated bookmarks and tags.</li> <li>3. She can delete a bookmark or tag from a user.</li> <li>4. If she considers a user account is only made of spam, she can delete all together.</li> <li>5. The selected entries are erased from the system.</li> </ol> |  |
| <b>Use cases Related to Primary Path:</b> U2.  |  |
| <b>Alternative Path:</b> 2.1 If the bookmark to delete has been saved by more than 3 users a confirmation alert will be presented. If a tag is used by more than one user, the confirmation will appear too.   |  |
| <b>Use cases related to alternative:</b> None.   |  |
| <b>Exceptions:</b> None.   |  |
| <b>Use cases related to exceptions:</b> None.  |  |

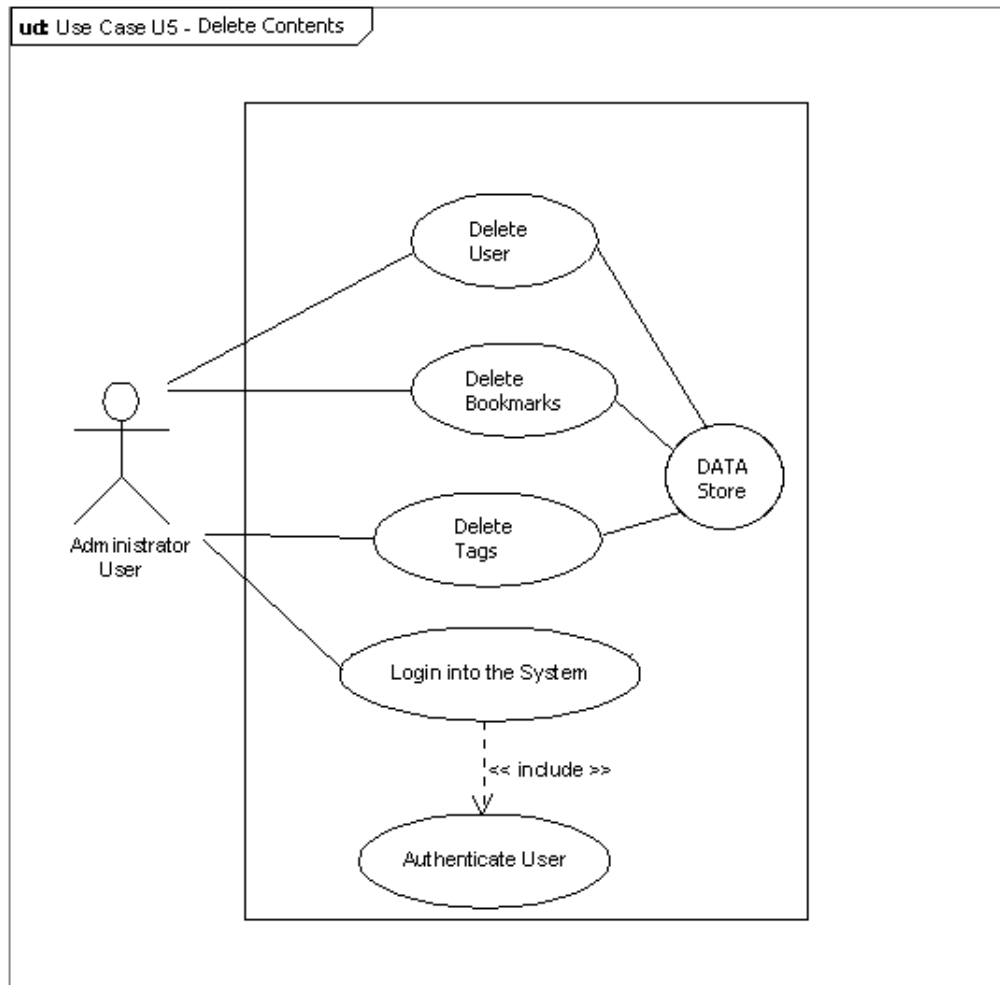


Figure 5.9: Use Case Number5: Delete Contents.

### 5.4.2 Class Diagrams

Class diagrams identify the class structure of a system, including the properties and methods of each one of the classes. On these diagrams normally the various relationships that can exist between classes such as inheritance are illustrated.

The main class diagram in the Figure 5.10 shows the most important classes of the Social Bookmarking System. There is a user superclass user with two subclasses: Registered User and Administrator. They share some common properties such as the name and the account inherited from his parent but they also implements some

methods of their own (for example the Administrator user implements has some unique methods such as `deleteAccount()`, `deleteBookmark()` and `deleteTag()`). The Registered User subclass also have its own methods such as `saveBookmark()` and `browseBookmar()`. She also inherits the name and account attributes from its parent class (the user class).

There is also another important superclass called `model` with different data properties. The bookmark, tag and link are subclasses and they also inherit from its model parent the different properties they can hold. The bookmark class send messages to the link and tag classes to store the information related to each one of the bookmarks.

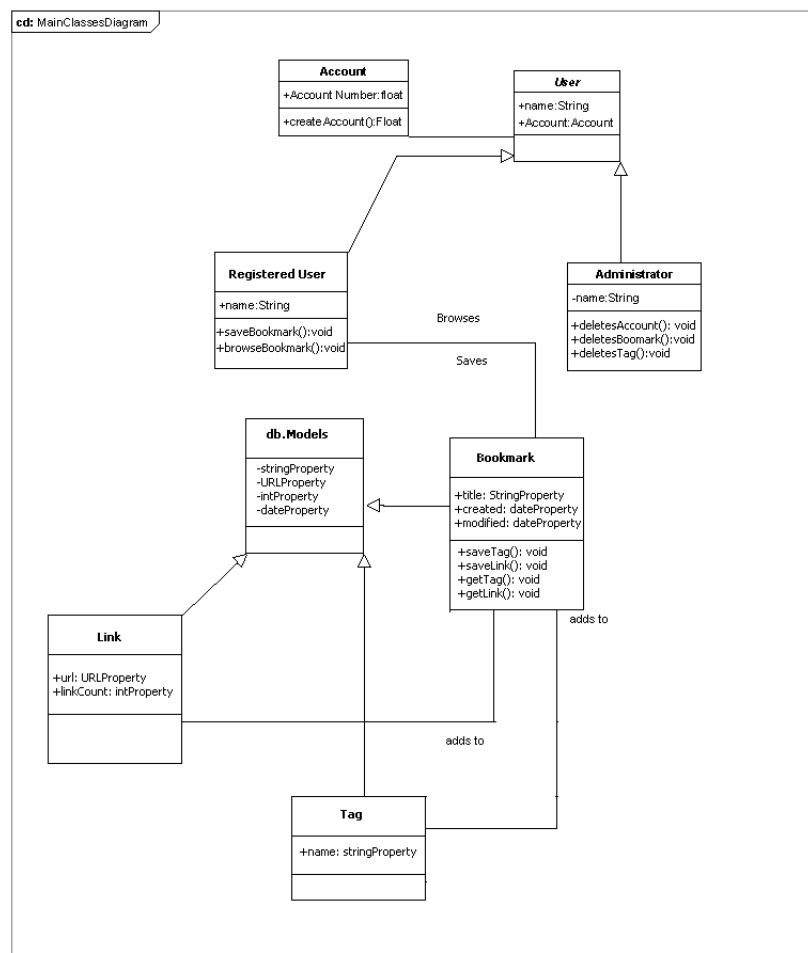


Figure 5.10: Main Classes Diagram.

### 5.4.3 Interaction Sequence Diagrams

The Interaction Sequence diagrams describe the interactions between the different classes composing a system to achieve a desired result. The message are the definition of the communication links between the different classes. On those diagrams the objects appear in the horizontal axis and the time in the vertical one.

#### **Add / Edit Bookmark**

The Add / Edit Bookmark diagram shows the interaction between the Registered User, Bookmark and Data Store objects. The Sequence begins when the user saves a bookmark. This message is sent to the Bookmark Object which can communicate with the Data Store object. It sends a message to adds the data into the bookmark repository. Now the Registered User object can send a message to edit a Bookmark that works in the same sequence, but instead of creating a new record into the bookmark repository. It also can send a message to delete a bookmark. Once the Bookmark object receives this message it will delete its contents and send a message to the Data Store object to remove the bookmark from the repository.

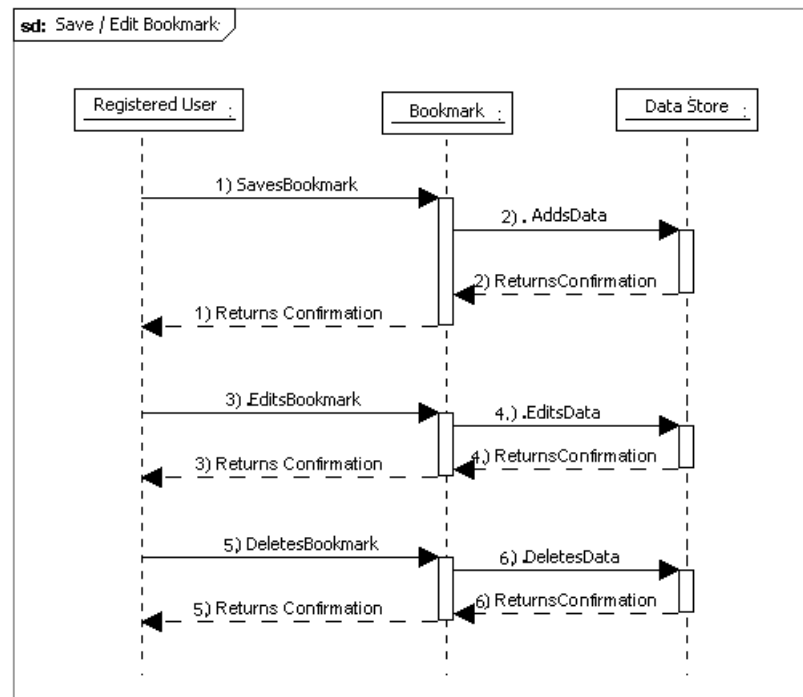


Figure 5.11: Interaction Sequence1 - Add / Edit Bookmark.

### Browse User Bookmark / Tag Bookmark

The Browse User Bookmark / Tag Bookmark diagram shows the interaction between the User, Bookmark and Data Store objects. The Sequence begins when the user browse a bookmark by user. This message is sent to the Bookmark Object which can communicate with the Data Store object. It sends a message to get data from the bookmark repository. The User object can also send a message to request the Bookmarks by tag. The process behaves in a similar sequence but the Bookmark object instead of requesting the bookmarks by user to the data store object is asks for the bookmarks associated with a particular tag. In both scenarios the data store object returns a message with the data matching the criteria and the bookmark object returns to the user the requested bookmarks.

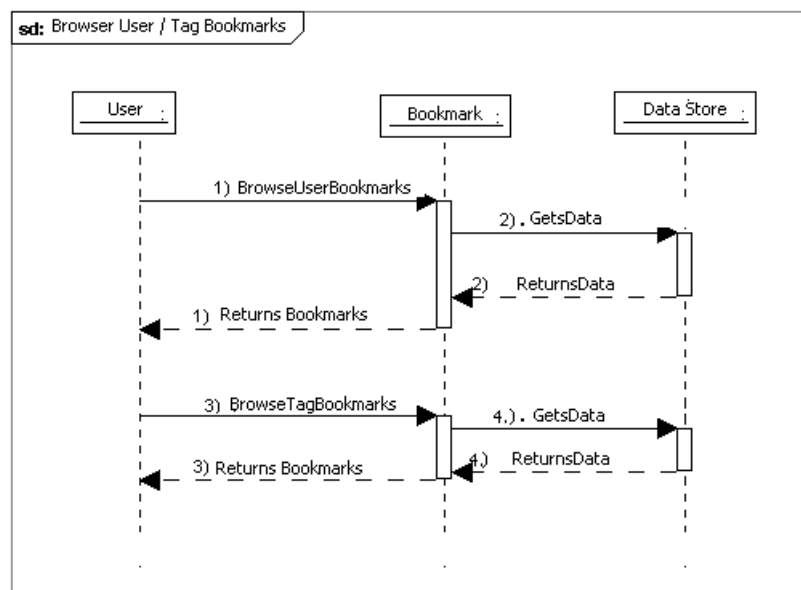


Figure 5.12: Interaction Sequence2 - Browse User Bookmark / Tag Bookmark.

## 5.5 Backend Design

### 5.5.1 Entities and Relationships

There are two concepts to take into consideration for designing the backend data-model of the Social Bookmarking System: *Folksonomy* and *Personomy*. As was discussed in a previous chapter the folksonomy is the a bottom-up the classification system generated by a community of users in particular system created by tagging or annotating a set of resources. The personomy is a subset of this Folksonomy taken into account only the metadata generated by one specific user. Based on these definitions we identify three type of elements into a Social Bookmarking sytem that are interrelated: The users, the bookmarks and the tags. Actually Hotho et al undertook a similar analysis in the definition of a formal model for folksonomies (Hotho et al., 2006b).

So they entities identified and the relationship between them are:

- **Users:** Each user can store one or more bookmarks into the systems with zero or more tags.
- **Bookmarks:** Each bookmark can be stores by one or more users with a different title and zero or more tags.
- **Tags:** Each tag can be associated to one or more bookmarks and can be used for one or more users.

We can also identify that there is a ternary relation between the Users, the Bookmarks and the Tags, where each resource entered into the system will be a combination of those three elements. The nature of this relationship must be considered where the entities and the relationships of the system are will the defined.

Some of of the attributes associated to each identity are defined in the following tables:



| Entity:User |         |
|-------------|---------|
| username    | varchar |
| password    | varchar |

Table 5.1: User Entity

| Entity:Bookmark |          |
|-----------------|----------|
| title           | varchar  |
| URL             | varchar  |
| date-created    | datetime |
| date-modified   | datetime |

Table 5.2: Bookmark Entity

| Entity:Tag |         |
|------------|---------|
| name       | varchar |

Table 5.3: Bookmark Entity

### 5.5.2 Entity Relationship Diagrams

Based on the ternary relationship described previously and the entities mentioned in the previous tables of the entities and relationship section we can produce a first entity relationship diagram illustrated in the Figure 5.13.

One of the problems arising here is the URL attribute of the Bookmark table. If two users or more users add a bookmark to the system with a different title but the same URL, the URL value will be repeated over and over, violating the rules of the first normal form (Codd, 1970). One solution to overcome this drawback is to separate this attribute to a new relation, so the same URL can belong to one or more bookmarks.

Therefore the redesigned entity relationship based on the previous analysis can be found in the Figure 5.14

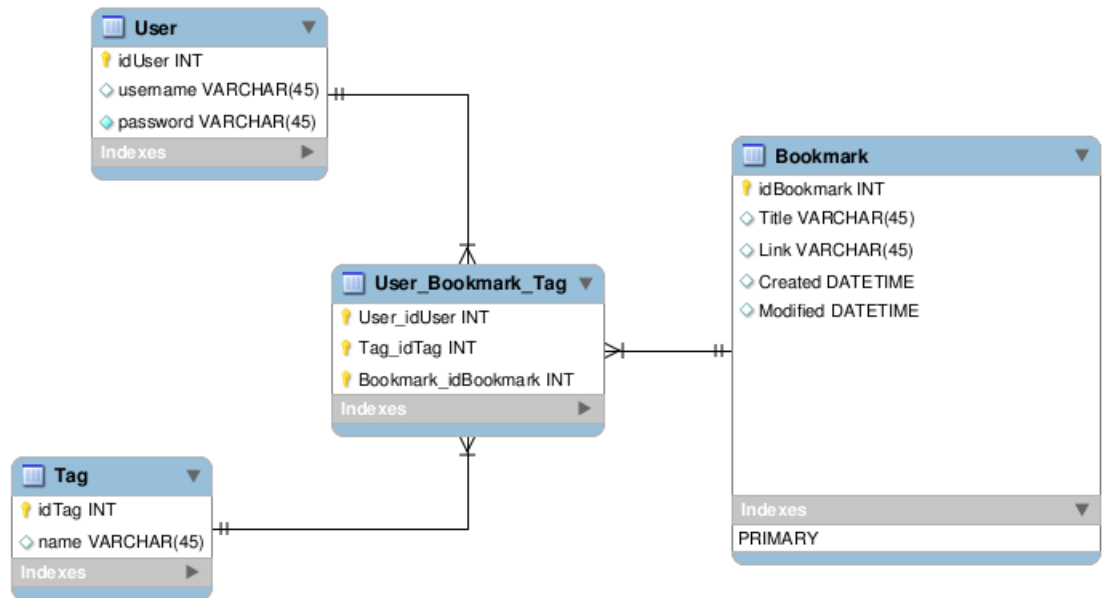


Figure 5.13: Entity Relationship Diagram - Draft1

## 5.6 Conclusions

The high level analysis performed at the beginning of the chapter helped to understand the main functionality of the system: it will be a web application where the registered users can store, share and retrieve their bookmarks or the ones stored by other users. Another decision taken after this analysis is the model-view controller-architecture to be used because the project is about a Web Application. The concept of Web Application was reviewed and it was decided to use the Django web application framework on the Google App Engine hosting platform. Finally the main use cases were modeled, the most important classes and interactions of the system were

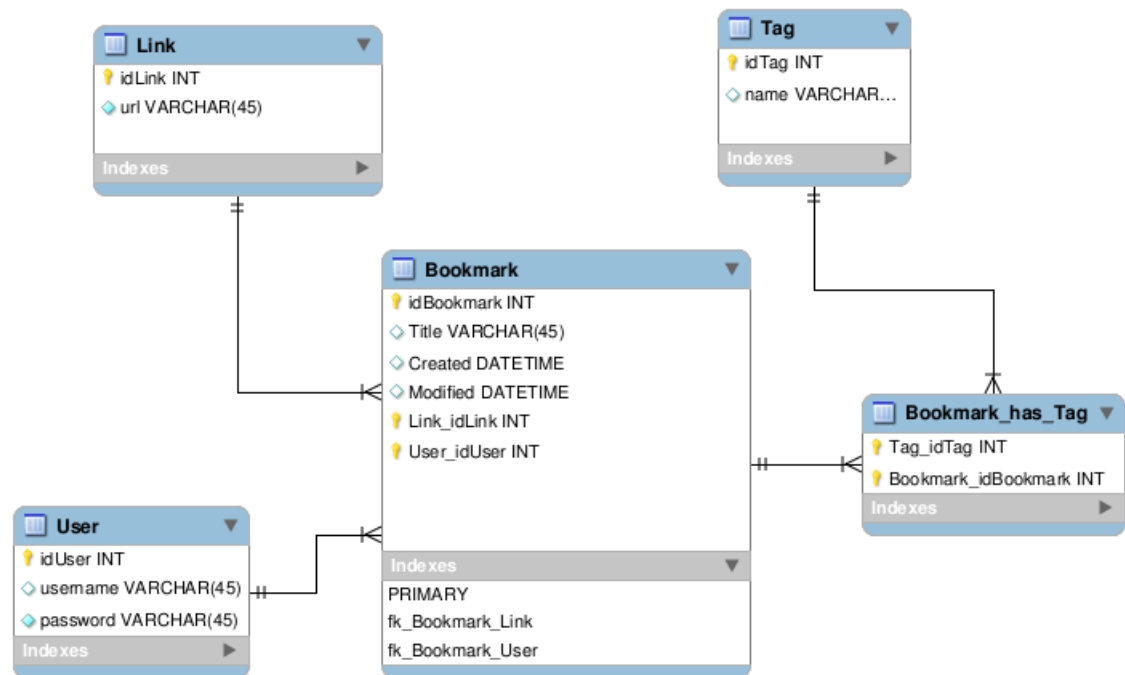


Figure 5.14: Entity Relationship Diagram for the Grays 2.0 Application (Final Version)

outlined using UML and the data model was planned and modeled using standard entity relationship diagrams so the next step now is to develop the web application.

# Chapter 6

## Development of Grays 2.0

*“I have the students learn Python in our undergraduate and graduate Semantic Web courses. Why? Because basically there’s nothing else with the flexibility and as many web libraries.”(Prof. James A. Hendler.)”*

### 6.1 Introduction

In the previous chapter the main use cases of the Social Bookmarking System were discussed and it was decided that the Django web application framework over the Google App Engine hosting services was going to be used since it was a good choice because Django has a good community of users, it implements the Model View Controller and it is based in Python, a programming language with a clear syntax and used in many agile projects. The task that will be explored in this chapter is how to implement the cases described in the previous chapter. The activities that will be covered in this chapter is how to set-up the development environment, how to install and use the Google App Engine development framework, how to create a Django application, develop the different components of the Web Application and how bind it with the App Engine. Finally the user interface design will be created and finally the data-model explored in the previous chapter will be incorporated

into the application. At the end of this phase there will a working software running both in the local development platform an online in the App Engine hosting account.

## 6.2 Development Environment

The first step before starting the development phase was to set-up and configure the development machine and the development environment. A first strategic decision taken was to use on this project only open source applications and packages covered by the Open Source initiative: “[www.opensource.org/licenses/alphabetical](http://www.opensource.org/licenses/alphabetical)”. The main reason to make this decision was to avoid having to go over many different kind of EULAs (End User License Agreements) from each of the proprietary software packages and the second reason was to save on the costs of this project, since all the activities related to this project can be accomplished using those open source systems and applications.

### 6.2.1 Development Hardware

The hardware used for the development of the project was a Netbook Samsung NC10 with the specifications described in the Table 6.1:

| Development Hardware Specifications |   |
|-------------------------------------|---|
| Storage Media                       | 160 GB 2.5” SATA HDD  |
| CPU                                 | 1.6GHz Intel Atom N270  |
| Memory                              | 1 GB  |
| Display                             | 10.2” (25.9 cm) 1024x600 LED-backlit TFT LCD                                  |
| Connectivity                        | 10/100 Mbit Ethernet 802.11b/g wireless LAN Bluetooth 2.0+EDR 3 USB 2.0 ports |

Table 6.1: Hardware Specifications

## 6.2.2 Development Software

Different software packages were used for the development of the project but as it was mentioned before all of them were compliant with the licenses from the Open Source Initiative, starting by the operating system: A netbook was bundled with Microsoft Windows XP, but it was replaced by the Linux distribution Ubuntu 8.10 (the Intrepid Ibex) covered by the GNU General Public License. This Linux distribution was selected because it has the widest community of users (it is the most popular Linux distribution), it has a strong emphasis in the usability and since it is based on Debian it is very easy to install new software using the apt-get program to manage the packages. For example for installing Python on the system it is enough to issue the following command in the linux terminal `sudo apt-get install python`

The other packages installed and used in the development of this project are in the Table 6.2.

The Django and Google App Engine packages are the most important pieces of the architecture for the application so they will be discussed separately

## 6.2.3 Django Setup

Since Django is a Python framework it needs Python to be in the system before attempting to install the framework in the development system. To verify if Python is installed in the machine and check the version it is necessary to open the terminal and issue the command `python -V`. In the standard distribution of Ubuntu 8.10 Python comes installed by default so the response received was Python 2.5.2. Django works with any version between 2.3 to 2.6. Django can be downloaded for free from the Django project website ([www.djangoproject.com/download/](http://www.djangoproject.com/download/)).

The version 1.0.2 was downloaded in a compressed format

(file `Django-1.0.2-final.tar.gz`) to the Desktop folder and then it was uncom-

| Development Software                          |  |
|---|--|
| <i>Package</i>                                | <i>Functionality</i>   |
| Gedit 2.24                                    | Simple text editor with coding highlighting (Typing LaTeX Document)                                      |
| Geany 0.14                                    | Small code editor / IDE with color coding support (Development for coding in Python, Javascript and CSS) |
| Firefox 3.03                                  | Web browser. (Testing the application)   |
| Opera 9.64                                    | Web browser. (Testing the application)   |
| Texmaker 1.7                                  | Integrated Latex editor with dvi, ps and pdf modules (Editing Latex document and creating final paper)   |
| KBibTex 0.21                                  | bib manager (Managing bibliographic references)  |
| Gimp 2.61                                     | Image Manipulation Program (Creating and editing graphic elements for the UI of the application)         |
| Koulour Paint 4.14                            | Small illustration program (Creating small designs for the paper)  |
| ArgoUML 0.28                                  | UML Editor (Designing use cases, classes and interaction diagrams)                                       |
| KSnapshots 0.81                               | Screenshot Software (Creating screenshots for the paper)   |
| MySQL Workbench 5.1                           | Database Designer (Creating the datamodel and designing the ER diagrams).                                |
| Python 2.5.2                                  | Dynamic object-oriented programming language (Used by Django and the App Engine).                        |
| <b>Django 1.0.2</b>                           | Web Application Framework (Developing and testing the Web Application).                                  |
| <b>Google App Engine SDK for Python 1.2.1</b> | App Engine SDK (Developing and testing the Web Application)  |

Table 6.2: Software Specifications

pressed and installed using

```
tar xzvf Django-1.0.2-final.tar.gz
cd Django-1.0.2-final
sudo python setup.py install
```

To test if Django was properly a test project was created named `django-test`, using the command `startproject`. A transcript of the process is next:

```
juan@ubuntu:~$ mkdir django-test
```

```
juan@ubuntu:~$ cd django-test/  
juan@ubuntu:~/django-test$ pwd  
/home/juan/django-test  
juan@ubuntu:~/django-test$ django-admin.py startproject  
django-test  
Error: 'django-test' is not a valid project name. Please use  
only numbers, letters and underscores.  
juan@ubuntu:~/django-test$ django-admin.py startproject  
django_test  
juan@ubuntu:~/django-test$ ls  
django_test  
juan@ubuntu:~/django-test$ cd django_test/  
juan@ubuntu:~/django-test/django_test$ ls  
__init__.py  manage.py  settings.py  urls.py
```

When a new Django project is created, it creates by default the 4 files that appear at the end of the transcript. `__init__.py` is a standard file to tell the Python compiler the contents of this directory should be treated as a package, `manage.py` is used to interact with the project in different ways (for example starting the server or the interactive shell), `settings.py` is self explanatory and `urls.py` is used by the URL dispatcher to match the URLs in the http request with the corresponding view.

A handy feature of the Django framework is it comes with a testing web server embedded so just by typing: `python manage.py runserver` the testing server is started and will serve content in the localhost address (`http://127.0.0.1:8000/`):

```
juan@ubuntu:~/django-test/django_test$  
python manage.py runserver  
Validating models...  
0 errors found  
  
Django version 1.0.2 final, using settings  
'django_test.settings'  
Development server is running at  
http://127.0.0.1:8000/  
Quit the server with CONTROL-C.  
[01/May/2009 09:03:30] "GET / HTTP/1.1" 200 2063
```



A screen-shot of the standard page Django offers when the testing server is started can be seen in the Figure 6.1. Finally to create a new “test\_application”, the command startup has to be used:

```
juan@ubuntu:~/django-test/django_test$ python
manage.py startapp test_application
juan@ubuntu:~/django-test/django_test$ ls
__init__.py  __init__.pyc  manage.py  settings.py
settings.pyc  test_application
urls.py  urls.pyc
juan@ubuntu:~/django-test/django_test$ cd
test_application/
juan@ubuntu:~/django-test/django_test/test_application$
ls
__init__.py  models.py  views.py
```

When a new application is generated Django creates a directory with the name of the application and puts inside to files: `models.py`, where the data models will be defined (it will be used by the object-relational mapper to create the relations in the relational database management system) and `views.py`, where the requests will be processed and send back to the template manager system. Those files will be used later on in the development of the Social Bookmarking application.

### 6.2.4 Google App Engine Setup

Both Python and Django are running fine in the system so the next step in the development platform configuration process is to download and install the Google App Engine SDK (Software Development Kit) and test it. The SDK *“include a web server application that emulates all of the App Engine services on your local computer. Each SDK includes all of the APIs and libraries available on App Engine. The web server also simulates the secure sandbox environment, including checks for attempts to access system resources disallowed in the App Engine runtime environment”*

One of the advantages of the SDK is that includes some utilities to upload the

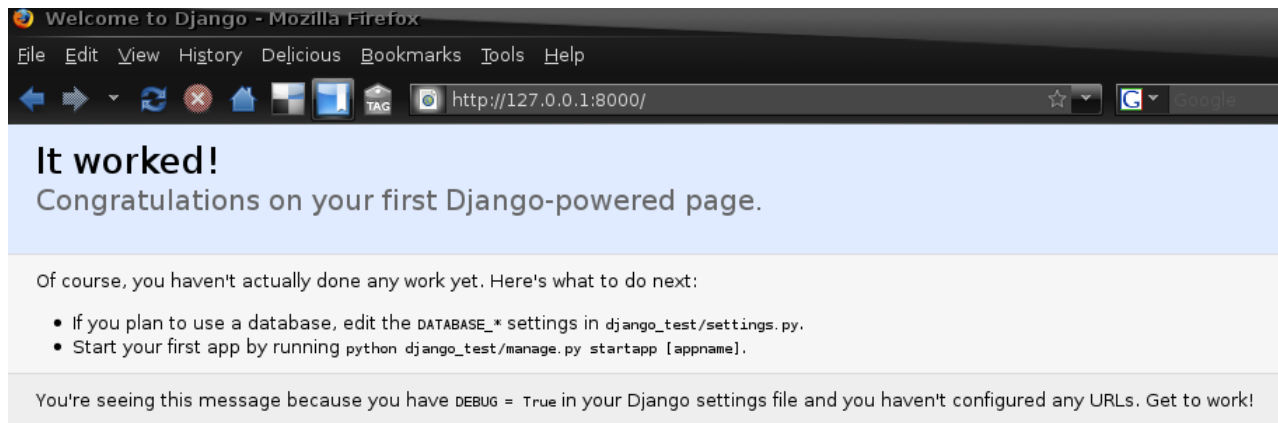


Figure 6.1: Django Testing Server - Welcome Screen

application to the Google App Engine platform; so once the application has been tested locally and it works it is very easy to synchronize the local version with the production environment. The process will be discussed later on in this chapter.

The Google App Engine Software Development Kit (SDK) can be downloaded for free from the official website (<http://code.google.com/appengine/downloads.html>). The file `google_appengine_1.2.1.zip` for the Linux platform was saved and uncompressed into the home folder. The site also offers a small “Hello World” sample (<http://code.google.com/appengine/docs/python/gettingstarted/helloworld.html>) very helpful to understand how the applications run in the Google App Engine and to test the development server. Each App Engine application needs an `app.yaml` configuration file that defines which scripts should handle the different URL requests. For the hello world example this file looks like:

```
application: helloworld
version: 1
runtime: python
api_version: 1
```

```
handlers:  
- url: /*  
  script: helloworld.py
```

The handlers version here tells the App Engine all the requests should be processed by the script `helloworld.py`. In this “Hello World” application. The script only prints this message.

```
print 'Content-Type: text/plain'  
print ''  
print 'Hello, world!'
```

A “helloworld” directory was created in the home directory of the development machine and the `app.yaml` and `helloworld.py` files were saved here. And the test server was loaded passing this directory as parameter:

```
juan@ubuntu:~/helloworld$ ls  
app.yaml  helloworld.py  
juan@ubuntu:~/helloworld$ cd ../google_appengine  
juan@ubuntu:~/google_appengine$  
./dev_appserver.py ../helloworld/  
INFO      2009-05-02 07:34:05,794 appengine_rpc.py:157]  
Server: appengine.google.com  
INFO      2009-05-02 07:34:05,818 appcfg.py:320]  
Checking for updates to the SDK.  
INFO      2009-05-02 07:34:06,303 appcfg.py:334]  
The SDK is up to date.  
INFO      2009-05-02 07:34:06,508 dev_appserver_main.py:463]  
Running application  
helloworld on port 8080: http://localhost:8080
```

The SDK test server loads the test web server in port:8080 by the application passed in the parameters. This “Hello World” application running can be seen in the Figure 6.2.

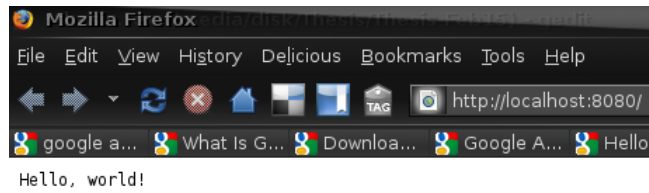


Figure 6.2: App Engine Hello World

## 6.3 Architecture

### 6.3.1 Django on Google App Engine

In the previous section the development environment set-up was described but to start developing the project one last step on setting this development environment to configure the Google App Engine to make sure it can run Django Applications on it. One of the advantages described previously is that since both Google App Engine and Django are designed to use the WSGI standard to run applications, the integration should be painless. Damon Kohler wrote a very useful article in the official app engine website about how to accomplish this task(Kohler, 2008). For using Django 0.96 (as it will be done in this project) the process is very simple: The `app.yaml` file is set-up in a mode where all the requests are passed to the `main.py` script:

Listing 6.1: `app.yaml` file

```
application: medical-bookmarks
version: 1
runtime: python
api_version: 1

handlers:
- url: /static
  static_dir: static
```

```
- url: .*  
  script: main.py
```

In the `main.py` script the WSGI handler imported from Django is defined and the logging information is passed to the App Engine (instead of the normal Django `manage.py`). A sample of this script that will be used for this project is available on the Google website:

Listing 6.2: `main.py` handler

```
import logging, os  
  
# Google App Engine imports.  
from google.appengine.ext.webapp import util  
  
# Force Django to reload its settings.  
from django.conf import settings  
settings._target = None  
  
# Must set this env var before importing any part of Django  
os.environ['DJANGO_SETTINGS_MODULE'] = 'settings'  
  
import logging  
import django.core.handlers.wsgi  
import django.core.signals  
import django.db  
import django.dispatch.dispatcher  
  
def log_exception(*args, **kwargs):  
    logging.exception('Exception in request:')  
  
# Log errors.  
django.dispatch.dispatcher.connect(  
    log_exception, django.core.signals.got_request_exception)  
  
# Unregister the rollback event handler.  
django.dispatch.dispatcher.disconnect(  
    django.db._rollback_on_exception,  
    django.core.signals.got_request_exception)  
  
def main():  
    # Create a Django application for WSGI.  
    application = django.core.handlers.wsgi.WSGIHandler()
```

```
# Run the WSGI CGI handler with that application.
util.run_wsgi_app(application)

if __name__ == '__main__':
    main()
```

Finally the normal `settings.py` normally created by Django for a new project has to be tweaked so the Django models will not be used (instead the App Engine ones will be used), the authentication an admin have to be disabled too (handled by the App Engine as well) and the sessions. The sample listing that will be adapted to this projects is:

Listing 6.3: settings.py script

```
import os
ROOT_URLCONF = 'urls'
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    # 'django.contrib.sessions.middleware.SessionMiddleware',
    # 'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.middleware.doc.XViewMiddleware',
)

INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    # 'django.contrib.sessions',
    'django.contrib.sites',
)

ROOT_PATH = os.path.dirname(__file__)
TEMPLATE_DIRS = (
    # Put strings here, like "/home/html/django_templates" or
    # "C:/www/django/templates". Always use forward slashes,
    # even on Windows.
    # Don't forget to use absolute paths, not relative paths.
    ROOT_PATH + '/templates',
)
```

Using this configuration, when the application is running and a URL is requested, the App Engine passes the request to `main.py` and since this file is creating the Django application for WSGI, the request will be handled by the URL dispatcher

as in any normal Django application. The system is ready to run a Django application on the Google App Engine, so the next step now is to edit the Django files composing the project and develop each one the parts of Medical Bookmarking System.

### 6.3.2 Grays2.0 Architecture

Since Grays 2.0 will be a Social Bookmarking system developed in Django over the Google App Engine, the architecture of the application will have elements of both as it was explained in the previous section. Since Django implements the MVC architecture and the presentation is separated from the logic, this project will use this implementation.

A typical Django application contains the following files:

- A `urls.py` file used by the URL dispatcher to identify view is called for any given URL pattern. This project will use that file.
- A `views.py` file that contains the handlers with the business logic for each one of pages requested. Grays 2.0 will use this file constantly since most of the development will take place here.
- A `models.py` file with a description of the database tables as Python classes. Since this project runs on the App Engine it will also have a `models.py` file but instead of having a map to a database it will map the Google Datastore.
- A `settings.py` file with the configuration of the application (it will be used as previously described)
- A `/templates/` directory where all the `.html` templates are stored and used by the template manager system. The Grays 2.0 project will create the `.html` files here.

Additionally Django offers an elegant module to handle forms (`forms.py`) that will be used. Also given the fact this is an App Engine application it needs to have the `app.yaml` and `main.py` described in the previous section. At the end of the development time the application source code files structure will look as:

```
|-- app.yaml
|-- forms.py
|-- main.py
|-- models.py
|-- settings.py
|-- static
|   |-- icons.jpg
|   '-- styles.css
|-- templates
|   |-- sample.html
|-- urls.py
|-- views.py
```

A mapping between this structure and the diagram that was exposed during the Analysis and Design phase is shown in the Figure 6.3.

## 6.4 Development Process

The files described in the previous sections were created in the application folder. The next step in the development process is to create the graphical interface the user will use to interact with the system as html templates. Those templates will be linked with the output from the views in the application. Finally the handlers will be coded.

### 6.4.1 GUI Design and Template development

The initial mock-ups of the Graphical User Interface design of the Medical Social Bookmarking, were designed using the GIMP image editor and then the Graphical elements were separated in different .PNG files and the rest of the structure of



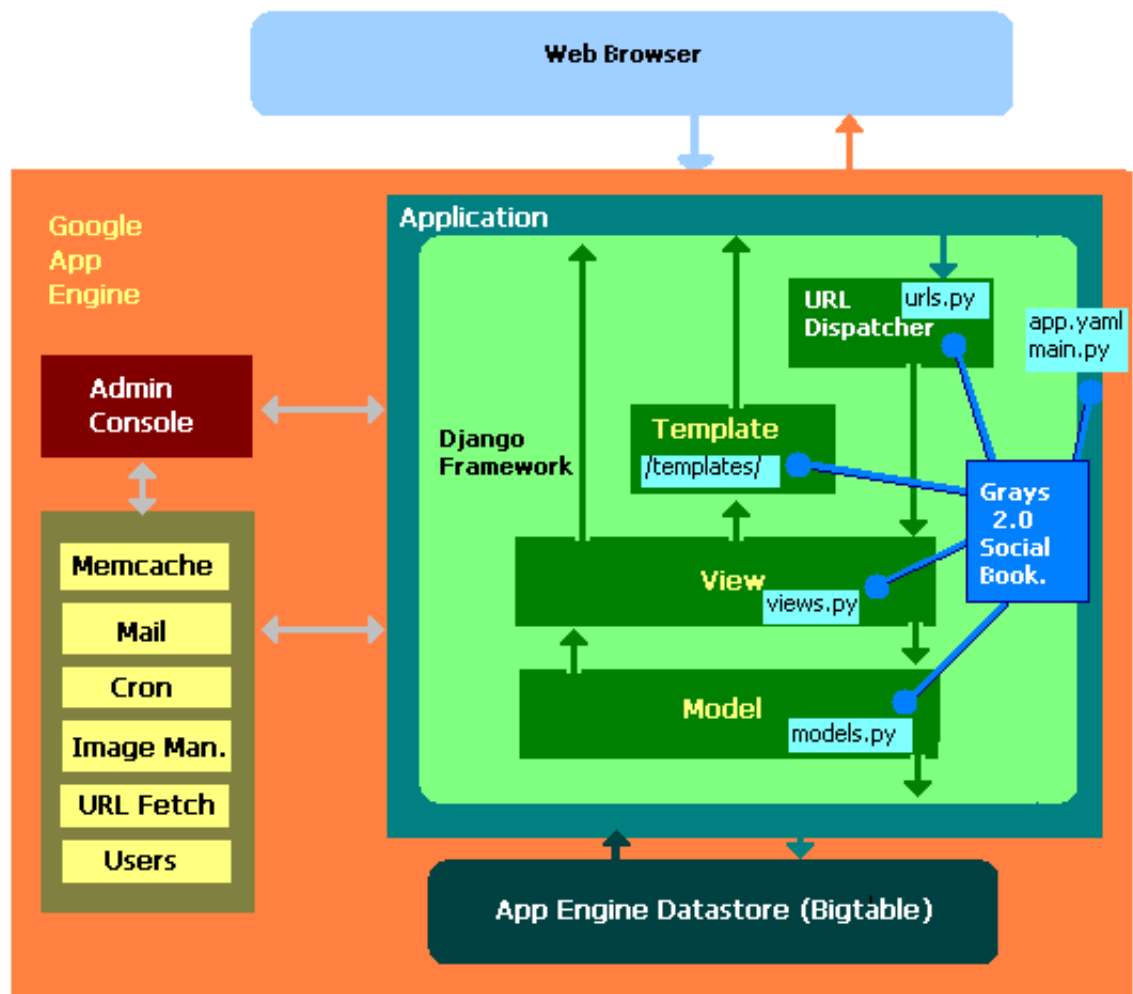


Figure 6.3: Grays 2.0 Architecture

the design was coded in HTML and CSS. The resulting files were stored in the `/template/` (html files) and `/static/` (css and graphic elements) directories of the project.

The objective of this phase was to create a generic template that could be used by all the different handlers of the application. Some general principles of Web usability were taken into account when designing the user interface, like the ones discussed in free online ebook published in “webstyleguide.com”. The aim of this process was to create a user friendly interface with a professional look, presenting the information in a clear way, putting the information in an position easy to find

and to give the user always possibility to either search for the information with a text-box or to browse the content by clicking around. The screen shot with the result of this process can be found in the Figure 6.4.



Figure 6.4: Generic Template

A very powerful feature of the Django template system is the concept of “Template inheritance” that works similar to the inheritance of the programming languages. The idea is the web developer can build a skeleton template with elements common to all the pages of the website and then define regions or blocks that can be overridden by the children templates extending from this parent. The template system also allows others to include additional templates inside a template, common in many Web scripting languages such as PHP or ASP. Therefore the generic template will be stored as a base template (`/base.html/`) with the variable “title” and “body” blocks that can be used by the templates extending it. The base template will also include some common templates for the header, the menu and the footer of the template.

Here is the code of the base template:

Listing 6.4: base.html template

```

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Medical Bookmarks - Juan Convers
    - {% block title %}Page Title{% endblock %}</title>
<link type="text/css" rel="stylesheet" href="/static/styles.css">

<script type="text/javascript">
var gaJsHost = (("https:" == document.location.protocol) ?
    "https://ssl." : "http://www.");
document.write(unescape("%3Cscript src='" + gaJsHost +
    "google-analytics.com/ga.js'
    type='text/javascript'%3E%3C/script%3E"));
</script>
<script type="text/javascript">
var pageTracker = _gat._getTracker("UA-9927**-*");
pageTracker._trackPageview();
</script>
</head>
<body>
<table width="700" border="0" cellpadding="0" cellspacing="0">

{% include "header.html" %}
{% include "menu.html" %}

<table class="white" width="100%" border="0" cellspacing="0"
    cellpadding="0">
    <tr>
        <td width="3%"></td>
        <td width="94%" valign="top" class="main">

{%block body%}{%endblock%}

{% include "footer.html" %}

</td>
</tr>
</table>
</body>
</html>

```

To test the templates in running in the application and determine if the inheritance was working a “generic” page was created without any content. This exercise

was very useful because the same steps will be followed in the development of each handler of the application. Those steps are:

1. Create an entry in the `urls.py` file with the URL pattern that will trigger the handler we are coding.

```
from django.conf.urls.defaults import *

urlpatterns = patterns(
    '',
    (r'generic', 'views.generic'),
)
```

2. Define and code the handler method in the `views.py` that will process the request and return the result with the template name that should be associated with this handler:

```
import django
from django import shortcuts

def generic(request):
    """Handler for showing a generic empty page in the
       Bookmarking System"""
    params = {}
    return shortcuts.render_to_response('generic.html', params)
```

3. Create and store the template in the `/templates/` directory that was referenced from the previous handler (`generic.html` in this case):

```
{%extends "base.html"%}
{% block title %}Generic Template{% endblock %}
{%block body%}
<h2>Generic Template</h2>

Put the content here.

{%endblock%}
```

In the local testing server, browsing this page with the address:  
`http://localhost:8080/generic/` returned the same results of the Figure 6.4.

## 6.4.2 Handler Implementation

The templates have been designed and tested with a generic handler. The next step in this development process is to program the handlers for each one the URLs implementing the use cases described in the analysis chapter.

### Create a new account and login

An additional advantage of using the Google App Engine as a hosting platform is the integration with Google Accounts for user authentication. There is user API available that allows the developer to use the Google login information for granting the user access to the application and interacting with her so she does not need to create an additional account. Furthermore, if the user is already logged-in any Google Service, the application does not need to ask to the user to login again because the authentication information is already there. If it is not the case the application can redirect the user to the normal Google Login page and go back to the application with user account already in place (see Figure 6.5 to have an idea how the process takes place)

A big advantage of this approach for the Social Medical Bookmarking application is that the developer do not need to write a single line of code for creating and managing the accounts. This part is taken care by the normal account creation process of Google, that includes a CAPTCHA challenge, to ensure that the registration is not automatically generated by a bot.

Another side benefit very important for the objective of this project is the reduction of the potential spam in the system, since a week account creation system could open the door to robots created by the spammers that can create accounts

**Medical Bookmarks uses Google Accounts for Sign In.**

Google is not affiliated with the contents of **Medical Bookmarks** or its owners. If you sign in, Google will share your email address with **Medical Bookmarks**, but not your password or any other personal information.

**Medical Bookmarks** may use your email address to personalize your experience on their website.

Sign in with your  
**Google Account**

Email:

Password:

☒ Remember me on this computer.

[I cannot access my account](#)

Figure 6.5: Medical Bookmarks - Google Login

and bookmark postings in bulk.

In order this functionality the application need to import the module ‘ ‘users’ ’ (from google.appengine.api import users).

It can create logging (users.create\_login\_url("/")) and

logout (users.create\_logout\_url("/")) pages.

A very useful method for always returning those parameters to the the templates was found in:

`http://google-app-engine-samples.googlecode.com/svn-history/r99/trunk/django_example/views.py.`

It is an open source project so it will be adapted to the views.py in this project with the following changes:

```
def respond(request, user, template, params=None):
    """Method to return a standard response."""

    if params is None:
        params = {}
    if user:
        params['user'] = user
        params['sign_out'] = users.CreateLogoutURL('/')
        params['is_admin'] = (users.IsCurrentUserAdmin() and
                              'Dev' in os.getenv('SERVER_SOFTWARE'))
    else:
```

```
params['sign_in'] = users.CreateLoginURL(request.path)
# Add the date by default in the return parameters
params['current_date'] = datetime.datetime.now()

# It always add the .html extension to the template name.
template += '.html'
return shortcuts.render_to_response(template, params)
```

### Add a new bookmark

The objective of this method as was discussed in the use case is to provide the user the interface to add a new entry to the bookmark repository, associated with a username and the number of arbitrary tags she decides. After registering the URL pattern in the `urls.py` file:

```
(r'^save/$', 'views.bookmark_save_page'),
```

A class is created in the `forms.py` using the Django forms module:

```
from django import newforms as forms

# Form for storing the Bookmarks.

class BookmarkSaveForm(forms.Form):
    url = forms.URLField(
        label='URL',
        widget=forms.TextInput(attrs={'size': 64})
    )
    title = forms.CharField(
        label='Title',
        widget=forms.TextInput(attrs={'size': 64})
    )
    tags = forms.CharField(
        label='Tags',
        required=False,
        widget=forms.TextInput(attrs={'size': 64})
    )
    edit = forms.CharField(
        required=False,
        widget=forms.HiddenInput()
    )
```

The next step is to write handler in `views.py`. First it will check that the user is logged in before continuing the process. If that's not the case the Django `http` module will be used for sending an error message:

```
from google.appengine.api import users
from google.appengine.ext.db import djangoforms

import django
from django import http
from django.http import HttpResponseRedirect, Http404

def bookmark_save_page(request):
    """Handler to add or edit a new bookmark in the DataStore """

    # Get the current user and if the user didn't sign in, it
    # will complain
    user = users.GetCurrentUser()
    if user is None:
        return http.HttpResponseForbidden('You must be signed in to
                                           add a new bookmark!!')

    ...
```

Now the handler has to check if the user is already posting information in the bookmark form. If that's not the case it will generate the form and it will pass it to the `respond` method, so it will be sent to the template.

```
if request.method == 'POST':
    ...
else:
    form = BookmarkSaveForm()
return respond(request, user, 'bookmark_save', {
    'form': form,
})
```

If the user is posting the bookmark information then the handler must check that the URL is valid, check if the link object already exists in the Datastore and



if that is the case to associate with the new bookmark created by the user. Before creating a new bookmark for this user for that particularly URL it has to check it does not exist (if it exists it means the user is editing the bookmark). Finally it has to link the set of tags with this bookmark and redirect the user to his bookmarks page. Note that the Bookmarks, Links and Tags objects are created based on the classes defined in “models.py”.

```
# In these couple of statements we import
# both the models an the forms

from models import *
from forms import *

if request.method == 'POST':
    form = BookmarkSaveForm(request.POST)
    if form.is_valid():

        # Create or get link
        url = form.clean_data['url']

        # Select if the link is already in the Datastore if that's
        # true it will use it otherwise it will create a new one.
        linkquery = db.GqlQuery("SELECT * FROM Link WHERE
                                url = :urlp", urlp=url)
        link2 = linkquery.fetch(1)
        if len(link2) == 0:
            link = Link(url=url, linkcount=1)
        else:
            if not request.has_key('edit'):
                link2[0].linkcount = link2[0].linkcount + 1
            link = link2[0]
        link.put()

        # Select if the bookmark is already in the Datastore if
        # that's true it will use it otherwise it will create a
        # new one.
        # Create or get bookmark.
        bookmarkquery = db.GqlQuery("SELECT * FROM Bookmark WHERE
                                    link = :linkp AND
                                    user = :userp",
                                    linkp=link, userp=user)

        bookmark2 = bookmarkquery.fetch(1)
        if len(bookmark2) == 0:
```

```

        bookmark = Bookmark(user=user, link=link,
                             title=form.cleaned_data['title'])
    else:
        bookmark2[0].title = form.cleaned_data['title']
        bookmark = bookmark2[0]

    # Create new tag list.
    tag_names = form.cleaned_data['tags'].split()
    for tag_name in tag_names:
        tag_name = tag_name.lower()
        tagquery = db.GqlQuery("SELECT * FROM Tag WHERE
                                name = :namep",
                                namep=tag_name)

        tag2 = tagquery.fetch(1)
        if len(tag2) == 0:
            tag = Tag(name=tag_name)
        else:
            tag = tag2[0]
        tag.put()
        if tag.name not in bookmark.taglist:
            bookmark.taglist.append(tag.name)
    bookmark.put()
    return http.HttpResponseRedirect('/user/%s/' % user)

```

A walk through of this process is demonstrated in the following set of screenshots (Figures 6.6 to 6.9):

### Search for bookmarks

The aim of this method is to give the user a search box where she can enter a query, search in the bookmark repository and return her a list of the bookmarks matching the query. Again here we need to register the URL pattern in the `urls.py` file:

```
(r'^search/$', 'views.search_page'),
```

Also a class is created in the `forms.py` using the Django forms module:

```

from django import newforms as forms

# Form for querying the bookmark datastore.
class SearchForm(forms.Form):
    query = forms.CharField(

```

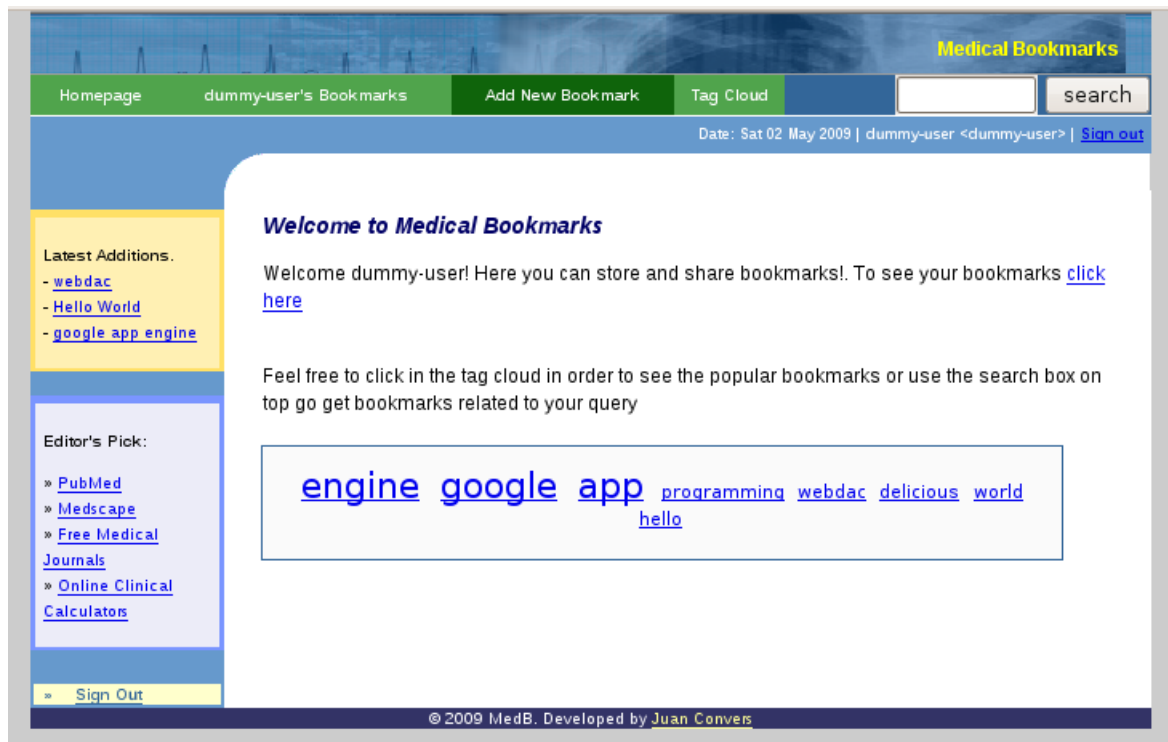


Figure 6.6: Add Bookmark - Menu item on the green bar

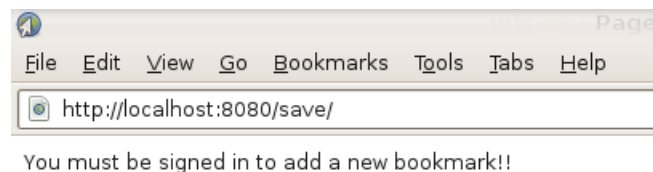


Figure 6.7: Add Bookmark - Error the user is not signed in.

```

label = '',
widget=forms.TextInput(attrs={'size': 32})
)

```

And a template `templates/search.html` that will show the search form or receive the results from the query. The capabilities of the Django Template Language are remarkable, since it accepts variables with values coming from the handler and

The screenshot shows the 'Add New Bookmark' form in the Medical Bookmarks application. The form is titled 'Add New Bookmark' and is located in the top navigation bar. The form fields are:

- URL:**
- Title:**
- Tags:**
- save** button

The form is displayed on a page with a sidebar containing 'Latest Additions' and 'Editor's Pick' sections. The top navigation bar includes links for 'Homepage', 'dummy-user's Bookmarks', 'Add New Bookmark', and 'Tag Cloud'. The page also shows a search bar and a date/time stamp: 'Date: Sat 02 May 2009 | dummy-user <dummy-user> | Sign out'.

Figure 6.8: Add Bookmark - The form to fill is printed

The screenshot shows the 'Bookmarks for user: dummy-user' page in the Medical Bookmarks application. The page displays the following information:

- Bookmarks for user: dummy-user**
- Sample Domain (1)** [edit] [delete]
- Created:** 2009-05-02
- Tags:** [sample](#) [domain](#) [test](#)
- Add a new bookmark** link

The page also features a sidebar with 'Latest Additions' and 'Editor's Pick' sections. The top navigation bar includes links for 'Homepage', 'dummy-user's Bookmarks', 'Add New Bookmark', and 'Tag Cloud'. The page also shows a search bar and a date/time stamp: 'Date: Sat 02 May 2009 | dummy-user <dummy-user> | Sign out'.

Figure 6.9: Add Bookmark - Bookmark saved and redirection to the user's page.

performs operations on them (for example “bookmark.title|escape”), it is able to test if a variable is present or not in the response (“if bookmarks”) and even do simple loops (“for bookmark in bookmarks’’):

```

{% extends "base.html" %}
{% block title %}Search Bookmarks{% endblock %}
{% block body %}

<h2>Search Bookmarks</h2>

Enter a keyword to search for:

<form id="search-form" method="get" action=".">
  {{ form.as_p }}
  <input type="submit" value="search" />
</form>

<div id="search-results">
  {% if show_results %}
    {% if bookmarks %}
      {% for bookmark in bookmarks %}
        <div class="greenmain">
          <a href="{{ bookmark.link.url }}" class="title">
            {{ bookmark.title|escape }}</a>
            ({{ bookmark.link.linkcount }})
            {% if user %}
              {% ifequal user bookmark.user %}
                <a href="/save/?key={{ bookmark.key }}"
                  class="edit">[edit]</a>
                <a href="/delete/?key={{ bookmark.key }}"
                  class="edit">[delete]</a>
              {% endifequal %}
            {% endif %}
          <br />
          {% if bookmark.created %}
            Created: {{ bookmark.created.date }}
          {% endif %}
          <br />

          {% if show_tags %}
            Tags:
            <ul class="tags">
              {% for tag in bookmark.taglist %}
                <li><a href="/kwd/{{ tag }}" />
                  {{ tag|escape }}</a></li>
              {% endfor %}
            </ul>
          <br />
          {% endif %}
          {% if show_user %}

```

```
        Posted By:
        <a href="/user/{{ bookmark.user }}" class="username">
            {{ bookmark.user }}</a>
        {% endif %}

    </div>
{% endfor %}

{% else %}
    <p>No bookmarks found.</p>
{% endif %}
{% endif %}
</div>

{% endblock %}
```

The last step is to write the code of the handler in `views.py`. The code is actually self explanatory: It will check if the request has a key “query” and if that is the case, it calls a search method on the Bookmarks object to get all the results matching the query. The models class has not been discussed yet (see next section) but it is important to clarify that bookmarks inherits from a “SearchableModel” and that is reason it supports the `search()` method directly.

```
from google.appengine.api import users

from models import *
from forms import *

def search_page(request):
    """Handler for searching the bookmarks """
    user = users.GetCurrentUser()
    form = SearchForm()
    bookmarks = []
    show_results = False

    if request.GET.has_key('query'):
        show_results = True
        query = request.GET['query'].strip()
        if query:
            # print query
            form = SearchForm({'query' : query})
            bookmarks = Bookmark.all().search(query).order("-created")
```

```

return respond(request, user, 'search', {
    'form': form,
    'query': query,
    'bookmarks': bookmarks,
    'show_results': show_results,
    'show_tags': True,
    'show_user': True
})

```

### Browse user bookmarks or tags

The purpose of this method is to browse the bookmarks created by the current user of the system or by other users. As was discussed in the Use Case section, the visitor users can also get the bookmarks created by any user. This method returns a list of the bookmarks created by the user selected in the URL. The first step as usual is to define the URL pattern in the `urls.py` file, but in this case we can use a powerful feature of the URL dispatcher in Django: It allows the developer to use regular expressions and this value is passed to the handler, so in our case it will be:

```

(r'^user/(.+)/$', 'views.user_page'),

```

The next step is to develop the handler part in `views.py`. Note that the `username` parameter received here comes from the regular expression part in the URLs file, so basically it will contain whatever the user puts after the `/user/` part in the URL.

```

def user_page(request, username):
    """Handler for User Page in the Bookmarking System"""

    user = users.GetCurrentUser()
    try:
        email = username
        #+ "@gmail.com" Add this part to the appspot.com version
        userbookmarks = users.User(email)
    except:
        raise Http404('Requested user not found.')
    bookmarks = Bookmark.gql("WHERE user = :userp ORDER BY created

```

```

DESC",userp=userbookmarks)
return respond(request, user, 'user_page', {
    'username': userbookmarks,
    'bookmarks': bookmarks,
    'show_tags': True
})

```

One challenge faced during the development of this part was to show the bookmarks coming from a user since the handler was working fine in the local development environment, but once the application was live it was always giving an empty result set. One solution devised to overcome this issue was to add the “gmail.com” part to the user in the production version. This hack solved the issue.

For browsing the bookmarks by tag a similar implementation was used:

```

urls.py
-----

(r'^kwd/([^\\s]+)/$', 'views.tag_page'),

views.py
-----

def tag_page(request, tag_name):
    """ Handler for showing a Tag page. """
    user = users.GetCurrentUser()
    tag = db.GqlQuery("SELECT * FROM Tag WHERE name = :namep",
                      namep=tag_name)
    tag2 = tag.fetch(1)
    # If it didn't find the Tag returns bookmarks as empty.
    # otherwise it query the datastore
    if len(tag2) == 0:
        bookmarks = ""
    else:
        for tag3 in tag2:
            tag = tag3.name
            bookmarks = Bookmark.gql("WHERE taglist = :tagp",tagp=tag)

    return respond(request, user, 'tag_page', {
        'bookmarks': bookmarks,
        'tag_name': tag_name,
        'show_tags': True,
        'show_user': True
    })

```



```
} )
```

## Tag Cloud

The last use case to be implemented in this part is the Tag Cloud. That means the possibility to browse the bookmarks visually using a cloud with the most popular tags highlighted by size according to their relevance. The Tag cloud will appear by default in the homepage and also there will be a deep URL (/kwd/). Both of those handlers called a `tag_cloud()` method that returns a dictionary with the tag cloud. Here is the implementation:

```
def tag_cloud():
    """Handler for generating Tag Cloud"""
    MAX_WEIGHT = 10
    temp_count = {}
    lower_count = 1
    higher_count = 1

    tagquery = db.GqlQuery("SELECT * FROM Bookmark")
    tagresult = tagquery.fetch(10000)

    # It iterates throught the tag list in the bookmarks
    # and for each element ofthe tag list it counts the
    # number of times it appears
    for taglist in tagresult:
        for uniquetag in taglist.taglist:
            if uniquetag not in temp_count:
                temp_count[uniquetag] = 1
            else:
                temp_count[uniquetag] = temp_count[uniquetag] + 1
                if temp_count[uniquetag] > higher_count:
                    higher_count = temp_count[uniquetag]

    tags = []

    # Calculate count range. Avoid dividing by zero.
    range = float(lower_count - higher_count)
    if range == 0.0:
        range = 1.0
```

```

# Calculate tag weights and assigns the values to the
# tags list of dictionaries that will be sent to the template.
# TODO: TO SORT ALPHABETICALLY THE DICT BEFORE.
for key in temp_count:
    temp_dict = {}
    weight = int(
        MAX_WEIGHT * (temp_count[key] - lower_count) / range
    )
    if weight < 0:
        weight = weight * -1
    temp_dict['name'] = key
    temp_dict['weight'] = weight
    tags.append(temp_dict)

return tags

```

And for tag cloud html templates part if both the homepage and the tag cloud page, and the CSS bit associated:

```

<table summary="tag cloud" class="bluebox" cellspacing="0">
  <tr>
    <td class="blueboxBody">
      <div id="tag-cloud">
        {% for tag in tags %}
          <a href="/kwd/{{ tag.name }}/"
            class="tag-cloud-{{ tag.weight }}">
            {{ tag.name }}</a>
        {% endfor %}
      </div>
    </td>
  </tr>
</table>

static/styles.css
-----

#tag-cloud a {
  margin: 0 0.2em;
}

.tag-cloud-0 { font-size: 100%; }
.tag-cloud-1 { font-size: 110%; }
.tag-cloud-2 { font-size: 120%; }
.tag-cloud-3 { font-size: 130%; }

```

```
.tag-cloud-4 { font-size: 140%; }  
.tag-cloud-5 { font-size: 150%; }  
.tag-cloud-6 { font-size: 160%; }  
.tag-cloud-7 { font-size: 170%; }  
.tag-cloud-8 { font-size: 180%; }  
.tag-cloud-9 { font-size: 190%; }  
.tag-cloud-10 { font-size: 200%; }
```

What is interesting about this implementation is that method counts the number of times each times appears and based on the frequency it gives to each one of them a weight value between 1 - 10 and then in the template with CSS, each one of the tags is shown with its relative size in comparison with the other tags of the system.

## 6.5 Database

One of the main differences between the Google App Engine platform regarding the storing the data is that it does not use a relational database management system (RDBMS) in the back-end but instead its own storing engine called the Google Datastore (based on on BigTable non-relational database). As was discussed before BigTable is a compressed, high performance, and proprietary database system that runs over on Google File System (GFS)(Chang et al., 2008). Some of the advantages of this model are the Datastore can execute multiple operations in a single transaction, can scale very well to manage large data sets and can optimize the way the data is stored so querying the data can be done very efficiently. Nevertheless one of the challenges faced by the developers of the traditional web applications with a relational database mindset is they need to change the way they model the data because the ‘JOIN’ operations as such as not supported. The Google Datastore is not a relational database. A web application developed for the Google App Engine has to store the data using the “Model” class. The process of instantiating this class is described in the official documentation as: *“An application defines a data*

model by defining a class that subclasses *Model*. Properties of the model are defined using class attributes and *Property* class instances. An application creates a new data entity by instantiating a subclass of the *Model* class. Properties of an entity can be assigned using attributes of the instance, or as keyword arguments to the constructor.”(Google.com, 2008a). Doing the analogy to the relational databases systems, the data “models” are equivalent to the tables and the “properties” are equivalent to the columns. Some of the properties supported by the Datastore used in this project are(Google.com, 2008b):

| Google App Engine Supported Properties |                            |  |
|--|----------------------------|--|
| Property class                         | Value type                 | Sort order   |
| StringProperty                         | str, unicode               | Unicode  |
| IntegerProperty                        | int, long                  | Numeric  |
| DateTimeProperty                       | DateProperty, TimeProperty | Chronological  |
| StringListProperty                     | list of a supported type   | If ascending, by least element; if descending, by greatest element |
| UserProperty                           | users.User                 | By email address (Unicode)   |
| LinkProperty                           | db.Link                    | Unicode  |
| ReferenceProperty                      | db.Key                     | By path elements (kind, ID or name, kind, ID or name...)           |

Table 6.3: Google App Engine - Properties

One interesting property is the “Reference”, that can be used to create a many-to-one relationship between the model with the property and the model referenced by the property (this is similar to the foreign keys in a relational database system). Finally since the Datastore is not exactly a RDBMS, SQL can not be used to retrieve the information from the Datastore; instead google offers “GQL” a SQL-like language with similar syntax but with some restrictions (as mentioned before it does not support JOIN operations).

### 6.5.1 Data Model

If there was a Relational Database System in the back-end of this application, like MySQL, based on the Entity Relationship diagrams created in the analysis and design chapter, the SQL statements we would need to use to create the database table would be:

```
-- -----  
-- Table 'Link'  
-- -----  
CREATE TABLE 'Link' (  
    'idLink' INT UNSIGNED NOT NULL ,  
    'url' VARCHAR(45) NOT NULL ,  
    PRIMARY KEY ('idLink') )  
ENGINE = InnoDB;  
  
-- -----  
-- Table 'User'  
-- -----  
CREATE TABLE 'User' (  
    'idUser' INT NOT NULL ,  
    'username' VARCHAR(45) NULL ,  
    'password' VARCHAR(45) NOT NULL ,  
    PRIMARY KEY ('idUser') )  
ENGINE = InnoDB;  
  
-- -----  
-- Table 'Tag'  
-- -----  
CREATE TABLE 'Tag' (  
    'idTag' INT NOT NULL ,  
    'name' VARCHAR(45) NULL ,  
    PRIMARY KEY ('idTag') )  
ENGINE = InnoDB;  
  
-- -----  
-- Table 'Bookmark'  
-- -----  
CREATE TABLE 'Bookmark' (  
    'idBookmark' INT NOT NULL ,
```

```

    'Title' VARCHAR(45) NULL ,
    'Created' DATETIME NULL ,
    'Modified' DATETIME NULL ,
    'Link_idLink' INT UNSIGNED NULL ,
    'User_idUser' INT NULL ,
    PRIMARY KEY ('idBookmark') ,
    INDEX 'fk_Bookmark_Link' ('Link_idLink' ASC) ,
    INDEX 'fk_Bookmark_User' ('User_idUser' ASC) ,
    CONSTRAINT 'fk_Bookmark_Link'
        FOREIGN KEY ('Link_idLink' )
        REFERENCES 'mydb'.'Link' ('idLink' )
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
    CONSTRAINT 'fk_Bookmark_User'
        FOREIGN KEY ('User_idUser' )
        REFERENCES 'mydb'.'User' ('idUser' )
        ON DELETE NO ACTION
        ON UPDATE NO ACTION)
ENGINE = InnoDB;

-----
-- Table 'Bookmark_has_Tag'
-----

CREATE TABLE 'Bookmark_has_Tag' (
    'Tag_idTag' INT NOT NULL ,
    'Bookmark_idBookmark' INT NOT NULL ,
    PRIMARY KEY ('Tag_idTag', 'Bookmark_idBookmark') ,
    INDEX 'fk_Tag_has_Bookmark_Tag' ('Tag_idTag' ASC) ,
    INDEX 'fk_Tag_has_Bookmark_Bookmark'
        ('Bookmark_idBookmark' ASC) ,
    CONSTRAINT 'fk_Tag_has_Bookmark_Tag'
        FOREIGN KEY ('Tag_idTag' )
        REFERENCES 'mydb'.'Tag' ('idTag' )
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
    CONSTRAINT 'fk_Tag_has_Bookmark_Bookmark'
        FOREIGN KEY ('Bookmark_idBookmark' )
        REFERENCES 'mydb'.'Bookmark' ('idBookmark' )
        ON DELETE NO ACTION
        ON UPDATE NO ACTION)
ENGINE = InnoDB;

```

But as it was discussed before the Google App Engine does not support the Relational Database Systems, so the Google Datastore has to be use and the application needs to instantiate the Model class in order to store and access the data. Based on

this constraint the data model of the Medical Bookmark applications will be stored in the file ‘‘models.py’’ with the following structure:

```
from google.appengine.ext import db
from google.appengine.ext import search

class Link(db.Model):
    url = db.URLProperty(required=True)
    linkcount = db.IntegerProperty(required=True)

class Bookmark(search.SearchableModel):
    title = db.StringProperty(required=True)
    user = db.UserProperty(required=True)
    link = db.ReferenceProperty(Link)
    created = db.DateTimeProperty(auto_now_add=True)
    modified = db.DateTimeProperty(auto_now=True)

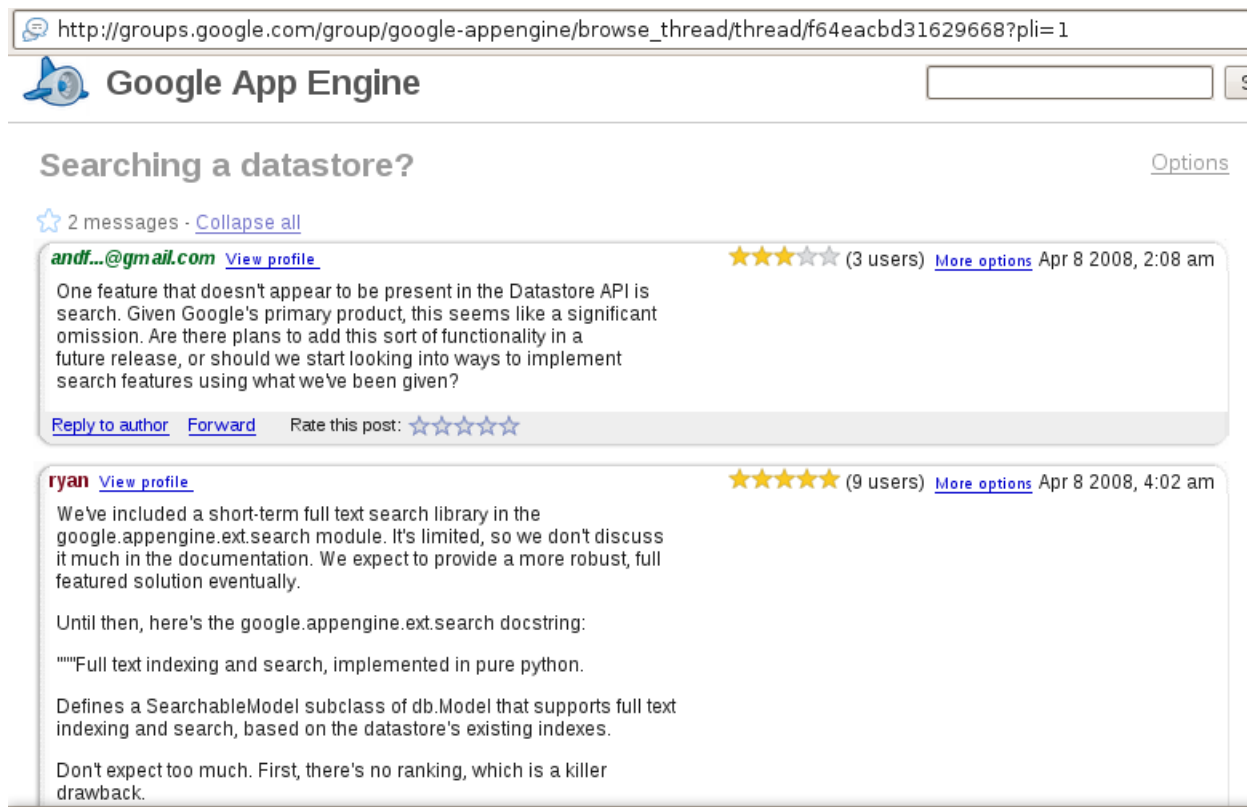
    # List with the Tags belonging to this user.
    taglist = db.StringListProperty()

class Tag(db.Model):
    name = db.StringProperty()
```

Finally as you can see in this model the Bookmark class is not inheriting directly from the `db.model`, but instead it inherits from `search.SearchableModel`. One of the issues faced during the development time was how to implement the search function of the application and the answer was found in one of the discussion forums (see Figure 6.10).

## 6.6 Uploading Application

The Google App Engine Python SDK is bundled with a command for interacting with App Engine called `appcfg.py`. This utility can be used to synchronize the new versions of the code with the production version of the application. In the <http://appengine.google.com/> address there is a console where a new appli-



The screenshot shows a web browser window with the address bar displaying a Google Groups URL. The page header includes the Google App Engine logo and a search bar. The main heading is 'Searching a datastore?' with an 'Options' link. Below the heading, it indicates '2 messages - Collapse all'. The first message is from 'andf...@gmail.com' (3 users, 4 stars) dated 'Apr 8 2008, 2:08 am'. The message text asks about a missing search feature in the Datastore API. The second message is from 'ryan' (9 users, 5 stars) dated 'Apr 8 2008, 4:02 am'. The message text provides a short-term full text search library and includes a code snippet for 'google.appengine.ext.search'.

http://groups.google.com/group/google-appengine/browse\_thread/thread/f64eacbd31629668?pli=1

**Google App Engine**

**Searching a datastore?** [Options](#)

☆ 2 messages - [Collapse all](#)

**andf...@gmail.com** [View profile](#) ★★★★★ (3 users) [More options](#) Apr 8 2008, 2:08 am

One feature that doesn't appear to be present in the Datastore API is search. Given Google's primary product, this seems like a significant omission. Are there plans to add this sort of functionality in a future release, or should we start looking into ways to implement search features using what we've been given?

[Reply to author](#) [Forward](#) Rate this post: ☆☆☆☆☆

**ryan** [View profile](#) ★★★★★ (9 users) [More options](#) Apr 8 2008, 4:02 am

We've included a short-term full text search library in the google.appengine.ext.search module. It's limited, so we don't discuss it much in the documentation. We expect to provide a more robust, full featured solution eventually.

Until then, here's the google.appengine.ext.search docstring:

```
"""Full text indexing and search, implemented in pure python.

Defines a SearchableModel subclass of db.Model that supports full text
indexing and search, based on the datastore's existing indexes.

Don't expect too much. First, there's no ranking, which is a killer
drawback.
```

Figure 6.10: Support Group - Search the datastore.

cation can be created under the appspot.com domain. For Grays 2.0 the sub-domain selected was “medical-bookmarks”, so the full URL of the application is `http://medical-bookmarks.appspot.com/`. The command used to upload each version to the hosting platform was:

```
appcfg.py update medical-bookmarks/
```

The first time the script uploaded the code it asked for the email and password of the Google Account. In the subsequent processes the information was not required (it is stored in a cookie). The information stored in the ‘‘app.yaml’’ file is used to identify the application identity (you can have up to 10 applications with each Google Account). That was the end of the process.



## 6.7 Conclusions

In the previous chapter the Analysis and Design was performed and some clear use cases were defined about what the Medical Social Bookmarking system was supposed to accomplish. In this chapter the development environment was created and two pieces of applications were installed on the development machine: The Django Web Application Framework and the Google App Engine SDK. Some test applications were coded and once everything was tested the actual development of the system was performed, creating the graphical templates and then developing all the use cases described in the previous chapter using Python as a language, Django as a framework and the App Engine as the testing server and hosting platform. Some of the challenges faced on this stage were the datamodel created in the design phase, since it had to be adjusted to meet the requirements of the Google App Engine Datastore, how to implement the search functionality, since it was not very obvious in the provided documentation and how to show the user's bookmarks once the application was deployed in the production server. The main output of this chapter is a first version of a fully functional Medical Bookmarking System Application running in the appspot.com domain. The next chapter will cover the testing phase of this product.

# Chapter 7

## Testing

*“To find the bugs that customers see - that are important to customers - you need to write tests that cross functional areas by mimicking typical user tasks. This type of testing is called scenario testing, task-based testing, or use-case testing.” (Brian Marick)*

### 7.1 Introduction

The process of reviewing the software and correcting the errors and bugs on it is normally an expensive procedure and can raise the costs of the development process. In a paper published by Westland in 2002 he found that the unsolved errors can increase the cost of the software development exponentially and the cost even increases more with each one the phases of the cycle. The conclusion of this study is that in depth testing should be performed before and after the system have been released(Westland, 2002). In the previous chapter the software development process was described and detail and the resulting product of the phase as a fully working piece of software with the use cases implemented. At this stage is very important to check this web application to make sure it does not contain bugs or critical errors that were not detected during the development phase. It is important

to use a systematic approach to the testing phase so the steps are executed in the same approach for each one of the cases. Fortunately the IEEE (Institute of Electrical and Electronics Engineers) published a document to be used in the software testing, the IEEE 829-1998 (829 Standard for Software Test Documentation)(IEEE, 1998). The document helps to define the different activities and processes related to the testing through a set of document such as the test plan, design specification, case specification, reports, etc. In this chapter a subset of this specification will be used to testing the Medical Bookmarks application.

## **7.2 Blackbox / Whitebox**

The Blackbox and Whitebox are terms used to designate different methods to perform the software testing. In blackbox testing the tester impersonates the external user who has no knowledge of inner-workings of the application and therefore it focus mainly on testing if the application meets the functional requirements. In this testing methodology the tester gives the application a set of valid and invalid inputs and determines if the application returns the expected results in each one of the cases. On the other hand whitebox testing requires for the tester with a good knowledge of the application internals (Risley, 2008). The main difference in this case is the tester know which classes will be instantiated, which methods will be called and which variables will be modified so when she enters an input the testing is done in each on the components involved in the tested functionality. Whitebox testing can be performed at different levels but the smallest component that can be tested it is called the unit testing. For this application a blackbox method will be used to emulate a real user (no knowledge of the inner workings).

## 7.3 IEEE 829 Testing

As discussed in the introduction the IEEE 829 provides a good framework for planning and executing the testing of a software package. In this project some of the documents from that specification will be used for conducting the testing. A test plan will be designed specifying how the testing will be done, what it is going to be tested and who will be performing the testing and its scope. Some test cases will also be defined, giving a general description, specifying the action to be executed, the expected result, the actual result and some additional notes.

## 7.4 Test Plan

The aim of this testing plan is to verify in Grays 2.0, the Medical Bookmarking System v1.0, implemented in this dissertation project meets the expectations set-up in the analysis phase and executes the different tasks implemented as expected. The tests are exclusively Black box (input/output), based on functionality and with no access to the code, just to the user interface. The tests will be divided in three, covering the authentication, add new bookmark, search for a bookmark, browse bookmarks by user or tag and tag cloud modules. A test case will be created for each one of the modules using the “test case” format from the IEEE specification; therefore the description, action, expected result, actual result and final test result will be recorded.

## 7.5 Test Cases

In this section the different test cases will be described and performed. The cases are divided in three subsections grouping the main functionalities of the system: Authentication, adding or editing bookmarks and browsing or searching for book-

marks. Again all this test cases will be conducted using a black box methodology. One prerequisite common for all the cases is to have an Internet connection and web browser. For certain test cases a Google account will be required.

### 7.5.1 Authorization

All the test cases in this section are associated with the authentication process. Here it will be tested if the user can login and logout in the system and see the menu options in the graphical interface, associated with each state.

#### User can login

|  |
|--|
| <b>Description:</b> A user can login into the system.  |
| <b>Prerequisites:</b> To have a valid Google Account   |
| <b>Steps to execute:</b> <ol style="list-style-type: none"> <li>1. The user goes to the homepage or any deep URL of the site and find the sign-in item on the menu.</li> <li>2. The user clicks in the sign-in link.</li> </ol>  |
| <b>Expected Result:</b> <ol style="list-style-type: none"> <li>1. Since the user is not in the system the “login” option should be shown.</li> <li>2. The user is provide his user name and password an submit the info.</li> <li>3. The system redirects to the homepage showing the user name in the interface.</li> </ol> |
| <b>Actual Result:</b> <ol style="list-style-type: none"> <li>1. The sign-in item menu was available in the homepage at the menu on the top, next to the date and on the left.</li> <li>2. After clicking in login, the system redirected the browser to the Google Accounts login.</li> </ol>                                |

3. Once the login information was provided the browser went back to the homepage of the bookmarking application.
4. In the homepage the name was printed in the welcome message and the logout option was available.

**Comments:** The authentication part was handled by the Google accounts system.

**Result:** Pass

### Registered user can logout

**Description:** A user logged in the the system can logout.

**Prerequisites:** To have a valid Google Account

**Steps to execute:**

1. The user first have to login (previous test case).
2. The user clicks in the sign-out link.

**Expected Result:**

1. Once the user is authenticated a “logout” option should be displayed in the interface.
2. When the user clicks in the logout option she should be logged out.

**Actual Result:**

1. The sign-out item menu was available only the user was logged in the system.
2. After clicking in the logout link, the user was logged out.
3. The user was redirected to the home page and the standard visitor page was printed.

**Comments:** The sign-out process was handled by the Google accounts system.

**Result:** Pass

**Logged user can not sign-in again and vice-versa**

|   |
|---|
| <b>Description:</b> A logged user can not login again and and logged out user can not logout again in the system.   |
| <b>Prerequisites:</b> To have a valid Google Account  |
| <b>Steps to execute:</b> <ol style="list-style-type: none"><li>1. The user click in sign-in item on the menu.</li><li>2. Once the user is in the system she tries to do it again.</li><li>3. The user click in sign-out item on the menu.</li><li>4. Once the user is out of the system she tries to do it again.</li></ol>   |
| <b>Expected Result:</b> <ol style="list-style-type: none"><li>1. Since the user is not in the system the “login” option should be shown.</li><li>2. Once she is registered the sign-in option should not be be available.</li><li>3. When the user logout the “sign-out” option should not be available.</li></ol>  |
| <b>Actual Result:</b> <ol style="list-style-type: none"><li>1. The sign-in item menu was available in the homepage at the menu on the top, next to the date and on the left. The “sign-out” option was not available.</li><li>2. After clicking in login, the system redirected the browser to the Google Accounts login.</li><li>3. Once the login information was provided the browser went back to the homepage of the bookmarking application and the “sign-in” option was replaced by “logout” option. It is not possible to login again</li><li>4. After clicking in the logout link, the user was logged out and redirected to the page with the “sign-in” option. The “Sign-out” option is not available anymore.</li></ol> |
| <b>Comments:</b> The authentication part was handled by the Google accounts system.   |
| <b>Result:</b> Pass   |

## 7.5.2 Adding or editing bookmarks

All the test cases in this section are associated with inserting information into the datastore, mainly by adding, editing bookmarks from the application. In this section it will be tested that only the registered user can add new information and that a registered user can add or edit new bookmark and the associated tags.

### Only registered users can add bookmarks.

|  |
|--|
| <b>Description:</b> A user has to be registered in the system before adding a bookmark.  |
| <b>Prerequisites:</b> None   |
| <b>Steps to execute:</b> <ol style="list-style-type: none"><li>1. The user makes sure she is not logged in.</li><li>2. The user clicks in the “Add New Bookmark” link.</li></ol>   |
| <b>Expected Result:</b> <ol style="list-style-type: none"><li>1. Since the user is not authenticated the system should not allow the bookmark to be added.</li></ol>   |
| <b>Actual Result:</b> <ol style="list-style-type: none"><li>1. As an unregistered user clicking the “Add New Bookmark” option in the system, the application printed a “You must be signed in to add a new bookmark!!”</li></ol> |
| <b>Comments:</b> Even the test case is fine, it would be useful to redirect the user to the login page once the message is printed.  |
| <b>Result:</b> Pass  |

### Registered user can add a bookmark.



|  |
|--|
| <b>Description:</b> A registered user is able to add a new bookmark to the repository.   |
| <b>Prerequisites:</b> Logged in the system   |
| <b>Steps to execute:</b> <ol style="list-style-type: none"><li>1. The user clicks in the “Add New Bookmark” link.</li><li>2. The user fills out the form.</li><li>3. The user clicks submit.</li></ol>   |
| <b>Expected Result:</b> <ol style="list-style-type: none"><li>1. Once the user fill out the form a clicks submit a new bookmark will be added to her repository.</li><li>2. She will be redirected to her page to see her bookmarks and the new bookmark should appear there.</li></ol>  |
| <b>Actual Result:</b> <ol style="list-style-type: none"><li>1. Once the “Add New Bookmark” link was clicked, the add bookmark form with the “Url”, “Title” and “Tags” was presented.</li><li>2. After filling all the fields were filled out and the form was submitted the application redirected the user his bookmarks page.</li><li>3. The bookmark that was just added appears on the list.</li></ol> |
| <b>Comments:</b> None.   |
| <b>Result:</b> Pass  |

**Registered user can edit a bookmark.**

|   |
|---|
| <b>Description:</b> A registered user is able to edit a bookmark she already has in the repository. |
| <b>Prerequisites:</b> Logged in the system and a bookmark already saved.                            |
| <b>Steps to execute:</b>  |

1. The user clicks on her bookmarks link (for example “test’s bookmarks”).
2. The user select one bookmark and click in “edit”.
3. The user edit the title or the tags and save the bookmark.

**Expected Result:**

1. Once the user clicks in edit a form will allow her to edit the field of the saved bookmark.
2. When the new information is saved the bookmark should be updated and the information reflected in the user’s bookmark page.

**Actual Result:**

1. Each one of the saved bookmarks has a “edit” link next to it.
2. Once the “edit” link is clicked a form is printed with the “Url”, “Title” and “Tags” pre-filled.
3. The title and the tags were changed and the form was submitted.
4. The changes are reflected into the user’s bookmarks page.

**Comments:** When editing a bookmark it would be useful to show the user a message informing that the application is in “Editing” mode.

**Result:** Pass

**Only valid URLs can saved as Bookmarks.**

**Description:** A registered user can only save valid URLs as bookmarks.

**Prerequisites:** Logged in the system.

**Steps to execute:**

1. The user clicks in the “Add New Bookmark” link.
2. The user fills out the form with an invalid URL
3. The user clicks submit.

**Expected Result:**

1. The system should check the information entered in the URL field is a valid bookmark.
2. If the information is not a qualified URL it should not proceed.

**Actual Result:**

1. When it was attempted to save a bookmark with a random string in the URL the system printed a “Enter a valid URL” message.
2. The form was printed again with the same information entered before.

**Comments:** It was noted that the URLs without the “http” protocol were consider invalid (technically correct). It would be more user friendly to automatic add the protocol if the user does not do it.

**Result:** Pass

### 7.5.3 Browsing or searching bookmarks

The test cases in part are related to the browsing and search functionalities of the bookmark system. It will test if a registered user can browse her own bookmarks. It also will be tested if the users can browse the bookmarks by username or by tag and if the systems allows to search bookmarks by entering a query.

Note all the following cases are based on the prerequisite that there are already two users on the system (“TestUserA” and “TestUserB”) and each one them has already save 2 bookmarks with the properties summarised in the following chart:

```
def user_page(request, username):
TestUserA
-----
Bookmark1
URL: http://www.test-a1.com
Title: Test Bookmark A1
Tags: test bookmark usera

Bookmark2
URL: http://www.test-a2.com
```

```

Title: Test Bookmark A2
Tags: test bookmark usera

TestUserB
-----
Bookmark1
URL: http://www.test-b1.com
Title: Test Bookmark B1
Tags: test bookmark userb

Bookmark2
URL: http://www.test-b2.com
Title: Test Bookmark B2
Tags: test bookmark userb

```

### Registered user can browse own bookmarks.

|   |
|---|
| <b>Description:</b> A registered user (TestUserA) can browse his own bookmarks.   |
| <b>Prerequisites:</b> Logged in the system with and the user already has two bookmarks saved.   |
| <b>Steps to execute:</b> <ol style="list-style-type: none"> <li>1. The TestUserA clicks in the “TestUserA’s Bookmarks” link.</li> </ol>   |
| <b>Expected Result:</b> <ol style="list-style-type: none"> <li>1. The system should check retrieve the bookmarks created by the user TestUserA .</li> <li>2. The boomarks should be displayed.</li> </ol>   |
| <b>Actual Result:</b> <ol style="list-style-type: none"> <li>1. When the TestUserA clicked on “TestUserA’s Bookmarks” link the browser open a page with the title “Bookmarks for user: TestUserA”.</li> <li>2. The page contains only the bookmarks created this user “TestUserA” (see Figure 7.1)</li> </ol> |
| <b>Comments:</b> None   |

**Result:** Pass



Figure 7.1: Registered user can browse own bookmarks.

User can browse bookmark by username.

**Description:** A user (TestUserA) can browse bookmark by username.

**Prerequisites:** Logged in the system. TestUserB has two bookmarks saved.

**Steps to execute:**

1. The TestUserA types in the URL “/user/TestUserB/” in the web browser.

**Expected Result:**

1. The system should check retrieve the bookmarks created by the user TestUserB
- .
2. The boomarks should be displayed to TestUserA.

**Actual Result:**

1. When the TestUserA type the URL in the Web browser a page was opened with the title “Bookmarks for user: TestUserB”.

2. The page contains only the bookmarks created this user “TestUserB” (see Figure 7.2)

**Comments:** The functionality works but it would be useful to have a page with a list of users or top users

**Result:** Pass

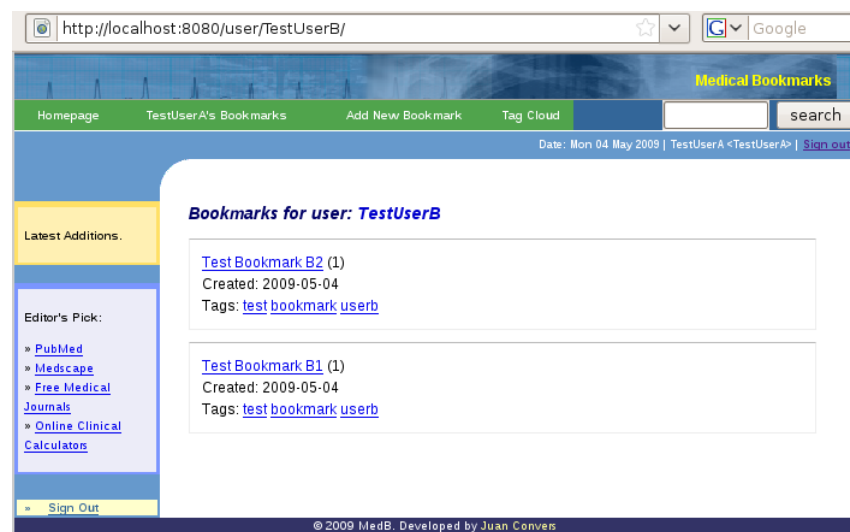


Figure 7.2: User can browse bookmark by username.

User can browse bookmark by tag.

**Description:** A user can browse bookmark by tag.

**Prerequisites:** There are four bookmarks saved with the tag “test”.

**Steps to execute:**

1. The user types in the URL “/kwd/test/” in the web browser.

**Expected Result:**

1. The system should check retrieve the bookmarks associated with the selected tag.

|   |
|---|
| 2. The bookmarks should be displayed, giving also the information about the creator.  |
| <b>Actual Result:</b><br>1. Once the URL was typed in the Web browser a page was printed the title “Bookmarks for tag: test”.<br>2. The page contains bookmarks created by both “TestUserA” and “TestUserB” |
| <b>Comments:</b> None   |
| <b>Result:</b> Pass   |

#### User can search for bookmarks.

|   |
|---|
| <b>Description:</b> A user can search the bookmark repository issuing a query.  |
| <b>Prerequisites:</b> There are has four bookmarks saved with the words “Test Bookmark” in the title.   |
| <b>Steps to execute:</b><br>1. The user types “Test Bookmark” and then “Test Bookmarked” in menu search-box.  |
| <b>Expected Result:</b><br>1. The system should search and retrieve the bookmarks matching the query in the title.<br>2. The bookmarks should be printed with the information of the creator.<br>3. It should return an empty set if there are no bookmarks matching the query. |
| <b>Actual Result:</b><br>1. When the query “Test Bookmark” was sent the application returned the bookmarks saved by both “TestUserA” and “TestUserB” .<br>2. When the query “Test Bookmarked” was issued the application returned an empty set.                                 |

|  |
|--|
| <b>Comments:</b> Even if the search engine works, it would be useful to display the user a “Not bookmarks found for this query” message if the query returns an empty set. |
|--|

|                     |
|---------------------|
| <b>Result:</b> Pass |
|---------------------|

## 7.6 Conclusions

At the beginning of the chapter it was discussed the importance of the testing process in any software development process since the testing permits to discover bugs and errors that can be very costly down the road once the application has been deployed. In the previous chapter the use cases from the analysis and design chapter were implemented and the outcome of that process was a first version of the functional software: A Medical Social Bookmarking application. The objective of this chapter was to create a testing plan so the different functionalities of the application could be tested to make sure it did not have major bugs and the main operations were working as expected. The execution of the test plan demonstrated the application is working and the users are able to save and search for bookmarks. There were some minor improvements suggestions that could be implemented in the next iteration but the core functionality of the Bookmarking Application was working as expected. Now the final stage of this project is to perform an evaluation on the application. This will be carried out in the next chapter.



# Chapter 8

## Evaluation

*“Correctness is clearly the prime quality. If a system does not do what it is supposed to do, then everything else about it matters little.” - Bertrand Meyer*

### 8.1 Introduction

Evaluation is defined by the the Evaluation Centre of Western Michigan University as the *“The systematic process of determining the merit, value, and worth of someone (the evaluatee, such as a teacher, student, or employee) or something (the evaluand, such as a product, program, policy, procedure, or process)”* (ec.wmich.edu, 2009). Every product should be evaluated to determine its relative value if compared with similar products or initiatives. In the software engineering field the evaluation of the quality of the final product is very important step since it should be perceived as a en excellent one by the customers, the stakeholders and end users. The product should be regarded as a software that accomplishes the task it is supposed to perform. A high quality software should be the goal in any software development project and it should be achieved to give the product a good head start in that market where is intending to compete.

One of the problems arising in the software quality assessment process is to agree in what exactly this term means, because the definition can be very subjective. Kan describes how there are popular views about the quality where it is seen as something ethereal that can not be measured and how sometimes can even be viewed as a synonym for expensive or luxurious (luxury items are described as high quality ones). Those views contrast with the professionals view about quality as the conformance to requirements and the fitness for use (Kan, 2003). In the software engineering field the term quality normally refers to the quality of the design (how good is the design of the project) and the quality of conformance (how well the product implements this design).

The International Organization for Standardization (ISO) also offers a useful definition of Quality (*“The totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs.”*), Assessment (*“An action of applying specific documented assessment criteria to a specific software module, package or product for the purpose of determining acceptance or release of the software module, package or product”*) and Software Quality (*“The totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied needs”*). Actually the ISO 9126 standard is defined and used for the evaluation of software quality.

In the previous chapter a set of testing methods were performed on the working on the Medical Social Bookmarking System. Now the the last stage of this development process is to apply some of the International Evaluation Standards of evaluation and do an assessment on the product’s quality based on the input received from external users.

## 8.2 International Evaluation Standards - ISO 9126

In the introduction it was discussed how the quality evaluation can be difficult in certain cases because human subjective factors can play a role determining the quality of a product and in this case a software product. There are International Organizations like the ISO (International Organization for Standardization) that promulgates industrial and commercial standards on a global scales. Actually ISO has published a standard the ISO 9126, that deals precisely with the evaluation of the software. This standards divides the process into four different parts covering: The quality model, the external metrics, the internal metrics and the quality in the use of the metrics(ISO/IEC Standard, 2003).



Figure 8.1: ISO 9126 Standard

This standard does not specify the requirements for software, but rather it gives a quality model that should be applicable to every type of software. One the

most useful part of this standard is this quality model since it defines the 6 main quality characteristics a software product should be evaluated against: Functionality, reliability, usability, efficiency, maintainability and portability. Each one of this characteristics have at the same time sub-characteristics that are summarised in the Figure 8.1 and explained in detail in the following table (adapted from [www.sqa.net/iso9126.html](http://www.sqa.net/iso9126.html)):

| ISO 9126 - Quality model |                    |  |
|--------------------------|--------------------|--|
| Characteristics          | Subcharacteristics | Definitions  |
| Functionality            | Suitability        | This is the essential Functionality characteristic and refers to the appropriateness (to specification) of the functions of the software.  |
|                          | Accurateness       | This refers to the correctness of the functions, an ATM may provide a cash dispensing function but is the amount correct?.   |
|                          | Interoperability   | A given software component or system does not typically function in isolation. This subcharacteristic concerns the ability of a software component to interact with other components or systems. |
|                          | Compliance         | Where appropriate certain industry (or government) laws and guidelines need to be complied with, i.e. SOX. This subcharacteristic addresses the compliant capability of software.                |
|                          | Security           | This subcharacteristic relates to unauthorized access to the software functions.   |
| Reliability              | Maturity           | This subcharacteristic concerns frequency of failure of the software.  |
|                          | Fault tolerance    | The ability of software to withstand (and recover) from component, or environmental, failure.  |
|                          | Recoverability     | Ability to bring back a failed system to full operation, including data and network connections.   |

|                 |                   |   |
|-----------------|-------------------|---|
| Usability       | Understandability | Determines the ease of which the systems functions can be understood, relates to user mental models in Human Computer Interaction methods.  |
|                 | Learnability      | Learning effort for different users, i.e. novice, expert, casual etc.   |
|                 | Operability       | Ability of the software to be easily operated by a given user in a given environment.   |
| Efficiency      | Time behaviour    | Characterizes response times for a given throughput, i.e. transaction rate.   |
|                 | Resource behavior | Characterizes resources used, i.e. memory, cpu, disk and network usage.   |
| Maintainability | Analyzability     | Characterizes the ability to identify the root cause of a failure within the software.  |
|                 | Changeability     | Characterizes the amount of effort to change a system.  |
|                 | Stability         | Characterizes the sensitivity to change of a given system that is the negative impact that may be caused by system changes.   |
|                 | Testability       | Characterizes the effort needed to verify (test) a system change.   |
| Portability     | Adaptability      | Characterizes the ability of the system to change to new specifications or operating environments.  |
|                 | Installability    | Characterizes the effort required to install the software.  |
|                 | Conformance       | Similar to compliance for functionality, but this characteristic relates to portability. One example would be Open SQL conformance which relates to portability of database used. |
|                 | Replaceability    | Characterizes the plug and play aspect of software components, that is how easy is it to exchange a given software component within a specified environment.                      |

Since this standard does not prescribe specific quality requirements but rather gives a general framework for evaluating the software quality based on the criteria previously discussed, some of the areas can be useful for the evaluation of this project and how it was used in the evaluation methodology it will be discussed next.

## 8.3 Evaluation Methodology

When all the phases from the software development process were completed, the Medical Social Bookmarking application was uploaded to the `appspot.com` top level domain and it was already fully working, the next step was to make an evaluation of the software. The methodology decided to evaluate was to get feedback from real users, so a feedback form was created using the Google Docs form feature. An email was sent to 30 different registered users asking them to fill out the form, three weeks before the end of project when this paper was reaching the end of the creation process. The poll was sent to users working both in the IT field, familiarised with the the World-Wide Web technologies and Web 2.0 social services, and to less savvy users working the health care field. The objective of this decision was to have a broad sample of user with a different degree of expertise.

The feedback form included eleven different questions with the objective to evaluate the main functionalities discussed in the use cases: Insert bookmarks into the bookmark repository and search or browse bookmarks from the repository. Some other questions were designed to get a profile of the users using of the system and finally there was a question to provide general feedback or suggest new functionality. The plan of the evaluation was to analyse the data gathered from the users and apply some of the parameters of the quality model from the ISO 9126 standard.

The questions were clustered in the following categories for the results analysis:

### **User profile** questions:

- Are you familiar with the concept Web 2.0?
- Are you familiar with the the term social bookmarking?
- Are you a user or have used the following services? (Facebook, Delicious, Linkedin, Blogger, Twitter, Couchsurfing)

- Do you work in the health care field?

**Add bookmarks** questions:

- Did you visit the <http://medical-bookmarks.appspot.com/> website?
- Did you add a bookmark into <http://medical-bookmarks.appspot.com/>
- If you answered yes, how easy was it to find the information?

**Browse or search bookmarks** questions:

- Did you visit the <http://medical-bookmarks.appspot.com/> website?
- Did you browse / search for information there?
- If you answered yes, how easy was it to find the information?

**General overview** questions:

- In general how do you rate the site If you rate the site <http://medical-bookmarks.appspot.com/>
- What functionality is missing in the service (and you would like to have)?

One of the challenges faced during this phase was the response rate to the solicited feedback. From the 30 request sent through email, there were only 14 replies in the form and the time was running up so a second strategy used by the author was to use all his exiting social network accounts and promote the feedback form. The status in Facebook was changed a couple of times promoting the service and the feedback, a message was posted in the Dublin group of the Couch surfing network and the status of the Gmail account was updated with the Url from the docs form. Some screen-shots of those efforts can be observed in the figure 8.2. The strategy was effective since at the end of the period 43 different rows of data were gathered in the feedback form and that is the sample used for the results discussed next.

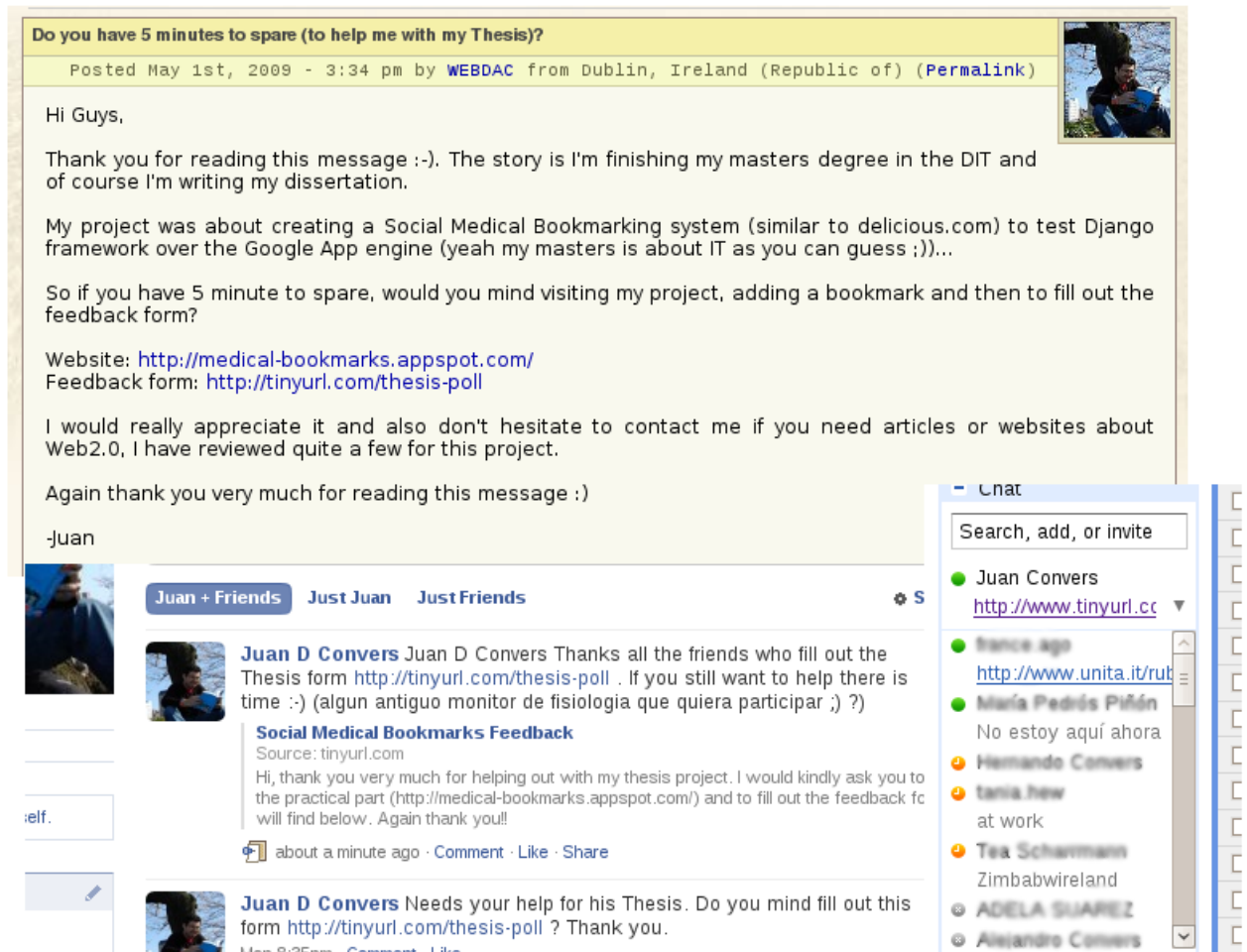


Figure 8.2: Use of social services to get feedback

## 8.4 Evaluation Results

As was discussed in the previous section a total of 43 forms were filled out using a Google docs form. The data was stored in a spreadsheet and exported as a CSV file. A Python script (included in the appendixes) was written to analyse the data and gather the statistics that will be discussed next.

### 8.4.1 Users Profile

One of the first conclusions about the user's profiles was that most of the have a good knowledge about the concepts related to Web 2.0 and Social Bookmarking



systems, since around 80% answered “Yes” to the questions about the familiarity with those concepts as can be observed in the figure 8.3. Some of the comments left on the areas for improvement field, gave some hints about the expertise of the users (ie. *“I would like to see a suggestion example when I am adding a Tag (e.g. should I use commas, do capital letters matters, etc.) in examples like <http://medical-bookmarks.appspot.com/user/luisellamz/> it is not very clear why there is a (1) besides the link and is missing an RSS feed :)”*)

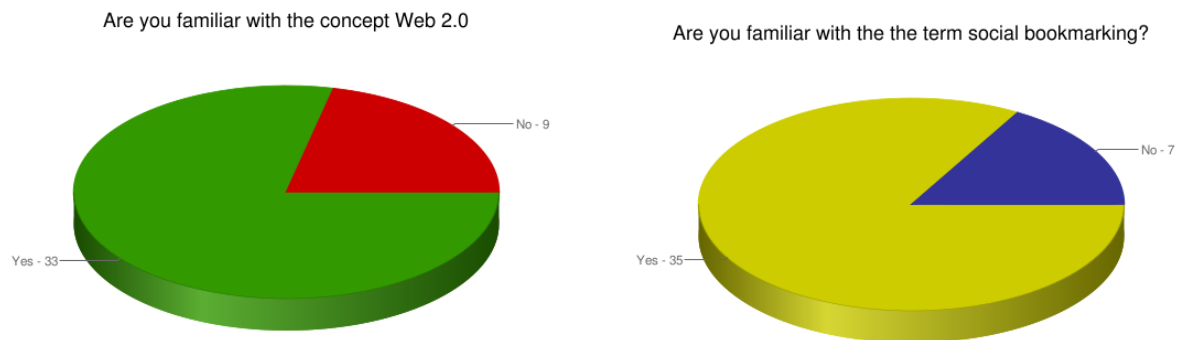


Figure 8.3: Familiarity with Web2.0 and Social Bookmarks

Another interesting data related to the profile of the users was the users were using or had used before. The overwhelming majority of users (more than 90%) were Facebook users, while between 40% and 50% were users of other services such as LinkedIn, Blogger or Twitter. Couchsurfing (specialized in the travelling and hospitality segment) had the lowest percentage usage with 20%. The Social Bookmarking service del.icio.us had a usage rate of 40% across the users participating in the feedback, which is good news because it means almost have the users had a previous reference point to compare Grays2.0 with. Finally another very important piece of data that has to be taken into account when analysing the data is the percentage of health care users, because only 16% belonged to the field. Apparently most of the users were IT professionals or people working in other areas. One of the reasons explaining this bias might be that the medical users were not too keen on trying out the product and give feedback. The profile of the users is summarised

in the graphs of the Figure 8.4.

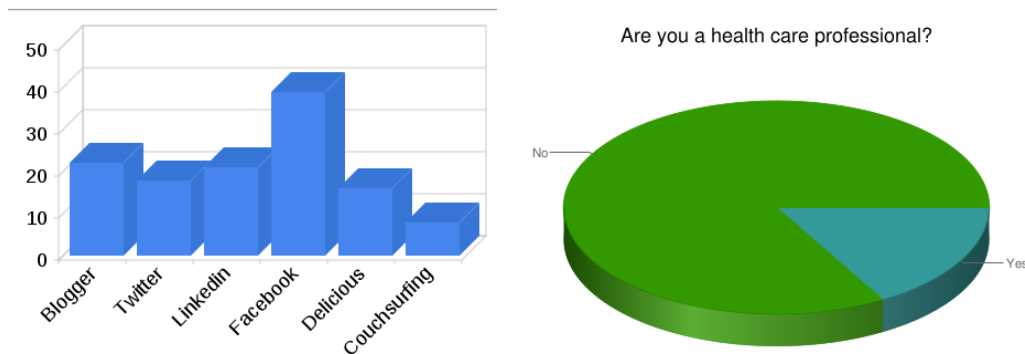


Figure 8.4: Social services and type of user.

### 8.4.2 Adding bookmarks

In the Figure 8.5 there is a summary about the data concerning the percentage of users that added a bookmarking into the repository and their subjective perception about how easy or difficult it was the process. The functionality, usability and efficiency parts of the quality model related to the “Adding” function were evaluated with those questions and with the open form at the end of the form. More than half of the users of the system 54% Added a new bookmark to the repository and 72% considered it was very or extremely easy to add a new bookmark (4 -5), only 8% (2 users) considered it was impossible to add a new bookmark. Based in this data it is considered that the functionality, usability and efficiency is very good. One of the reasons explaining the 8% of the impossible set is that the hosting server was returning back 500 errors during certain periods (ie. *“It returns a server error (500) so I was unable to use it. I would therefore say all possible functionality is missing.”*). Nevertheless since the application was back running and most the users were able to add bookmarks, the recoverability of the system is considered also good.

One of the comments entered in the improvements sections related to the usability was: *“Complete URL need to be entered (including the http and the www part), if*

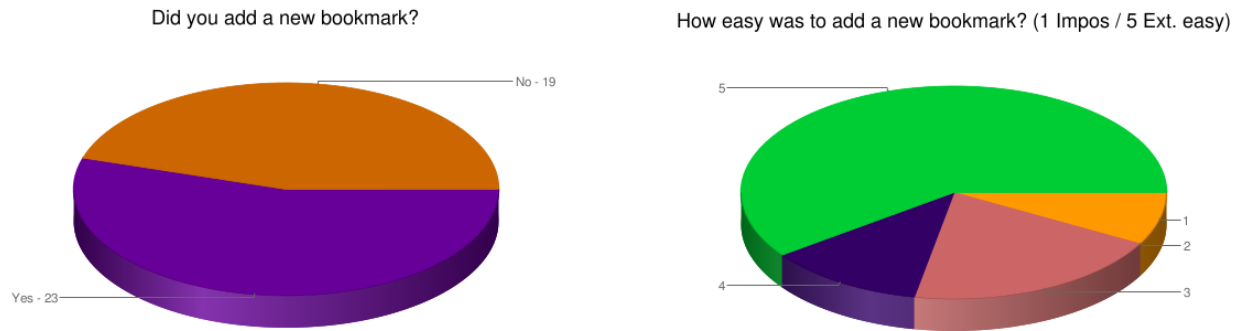


Figure 8.5: Add a new bookmark

*only the domain name could be entered, that can be good to have too.” and “Accepting different URL formats will be user friendly.”. In conclusion the evaluation of adding bookmarks was good but there are couple of areas of improvement.*

### 8.4.3 Searching of browsing for bookmarks

Most of the feedback left by the users in the open field at the end of the form was useful for evaluation the functionality of searching and browsing for bookmarks. Some of the quotes left by the users were:

*“It would be easy and handy if a bookmark has a description associated with it... like a brief what the bookmark is related to etc”, “maybe a map would be nice, as there are a lot of hospitals listed. that might make it easier for users to navigate. i assume, that most users are not looking for a particular hospital but rather for a hospital in the area within where they are living (eg. hospitals in Dublin). just an idea”, “tell users that there are zero search results, if that is the case”, “search even with misspellings”*

Nevertheless analysing the data from the questions related to browsing 73% of the users who browse a bookmark responded it was very or extremely easy to browse or search for a bookmark. Now there is a 23% of the users who rate the functionality as 3 / 5 and probably the quotes transcribed before belonged to the users on that group (the data is compiled in the Figure 8.6). The functionality

and usability of this part of the system is considered good based on the feedback provided by the users. The same issue of the the hosting server was returning back 500 errors during certain periods was present here but again the recoverability was assessed as good. There were no feedback about critical problems accessing the bookmarks so even if there is room for improvement the overall evaluation of this part is satisfactory.

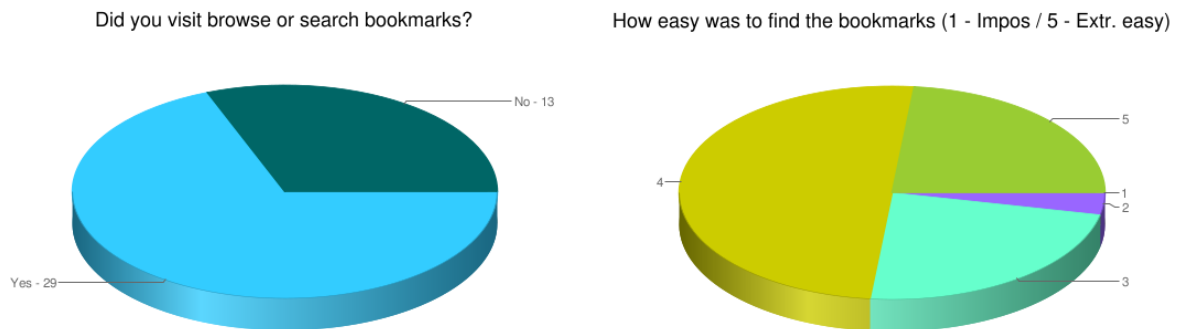


Figure 8.6: Add a new bookmark

#### 8.4.4 General Overview

The last question of the form before the open feedback was a wrap-up query about the general perception of the site in terms of quality, since the users were asked to rate the medical social bookmarking application from 1 to 5 (minimum - maximum) without giving them specific guidelines, so they took into account only their subjective perception. It was interesting to discover that half of the users rate the site with “4”, 17% rated as “5” and still some 28% of the users qualified the site with a “3”. This data is summarised in the Figure 8.7

Going back to the criteria discussed at the beginning of this chapter and based in all the points discussed briefly, the following point can be presented:

**Functionality:** The functions that were defined in the use cases, implemented in the software development and tested in the previous chapter were evaluated mostly

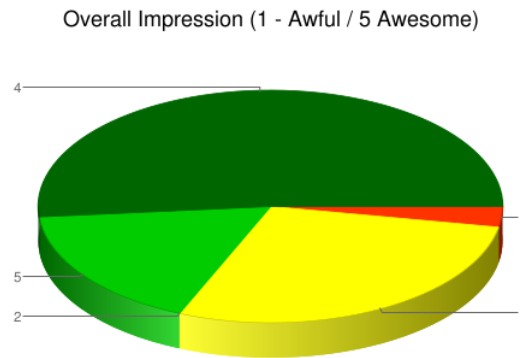


Figure 8.7: Add a new bookmark

positive by the majority of users. Most of the were able to add and browse bookmarks in the system. The users did not discuss it but the system was secure and only allowed the registered user to add new bookmarks, so the security is assesed as good.

**Reliability:** During the development and testing phase the application and the hosting platform demonstrated a good degree of reliability and no outages were notices. Nevertheless some users reported in the feedback form some 500 server errors that were reported. Now since the application was able to restart by itself and most of the time was up the recoverability is considered good as well.

**Reliability:** During the development and testing phase the application and the hosting platform demonstrated a good degree of reliability and no outages were notices. Nevertheless some users reported in the feedback form some 500 server errors that were reported. Now since the application was able to restart by itself and most of the time was up the recoverability is considered good as well.

**Usability:** Most of the users were able to perform the operations required like saving bookmarks and browsing bookmarks without any additional instructions, so the usability is evaluated as good too. They were some improvement suggestions to make it better that will be implemented in the next iteration but so far it works.

**Efficiency:** The application has not been used heavily yet but looking at the logs in

the Google App Engine console (see Figure 8.8) it seems the application is performing efficiently and is not spending many cycles of CPU, so this part is evaluated as good too.

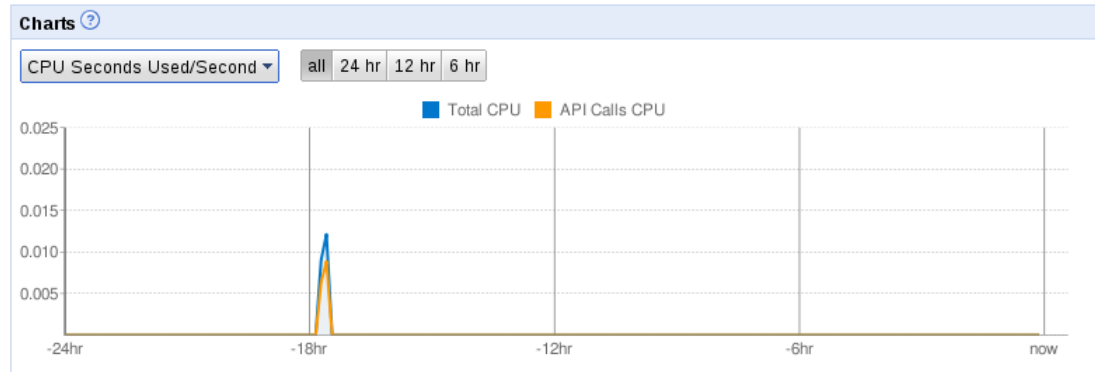


Figure 8.8: Appspot Dashboard CPU Usage.

**Maintainability:** This parameter was not evaluated by the users but given the fact the application was developed with a Web Framework that separates the business logic from the presentation, so far it has been very easy to maintain the application and correct small bugs. Therefore this parameter is evaluated as good too.

**Portability:** This areas has not been evaluated yet but in theory since the application was developed in Django it should be fairly easy to part the application to another hosting environment outside the Google App Engine, changing only the models and certain queries, so the portability should be good.

## 8.5 Conclusions

The last phase of this Software Development project about creating a Medical Bookmarking System using Django and running over the Google App Engine, after successfully doing the analysis, development and testing phases, was to evaluate the software. A group users from the system was selected to evaluate the system

through a form created using Google Docs. One of the conclusion from this methodology is that the social networks are powerful not only to keep in contact with the friends and colleagues but also to get feedback and they might be very useful in the future research projects. Just by changing the status in the different social network accounts the author was able to get additional responses and the dataset improved compared with the first request by email.

In this chapter it was also discussed the concept of quality, the subjectivity that might surround the term and how it is important to define standards so the software can be evaluated against them. The standard ISO 9126 was discussed since it is a very good framework for evaluating different kind of projects. Based on the objective and subjective feedback provide by the users, the Medical Bookmarking system was evaluated and it was discovered that most of adding and browsing function work, but also it was noticed that the users perceived there areas that can be improved in order to make the product more appealing to them (for example the graphical interface). In conclusion, the application is doing what it should do competently, but there areas where it could improve in order to be more successful and broaden its user base.

# Chapter 9

## Conclusions and Future Work

*“A conclusion is the place where you got tired of thinking.” - Arthur Bloch*

### 9.1 Introduction

The objective of this last chapter in this dissertation is about reviewing the work that has been carried out during the last months, including the software development process, writing of this paper, the critic evaluation of the findings and the analysis of the addition of this project to the body of knowledge and the impact it will have in the domain of the Web 2.0 applications created with the medical community in mind. A personal perspective about this project will be discussed and at the end of this chapter the ideas and suggestions for future work will be addressed as well.

### 9.2 Conclusions

#### 9.2.1 Main conclusion

One of main and most rewarding conclusions an the end of this projects is that the main objective was achieved. The main research aim was to the different applica-



tion frameworks and to select one to create a Web 2.0 Medical Social Bookmarking service that could be useful and interesting to the medical users. The Web frameworks were analysed and it was decided to develop the application using the Django MVC based application framework and to host it in the Google App Engine infrastructure to take advantage of the scalable platform they offer. Now at the time of writing this dissertation there is a fully functional web application running into the `medical-bookmarks.appspot.com` URL that has been used already by 60 different users and that offers the functionality of adding medical bookmarks and offers the possibility to browse and search for bookmarks. Even if some valid suggestions were provided during the evaluation phase that will be discussed later on in the section of future work, the basic functionality of adding and browsing bookmarks is working as expected so this main objective was achieved and this application could evolve and become a used service. The idea about having created a service that was not available before is gratifying and it is probably the most important conclusion of this work.

### 9.2.2 Additional Conclusions

During the development of the literature review, the analysis and design of the application, the development, testing and evaluation there some different conclusion that were drawn and discussed at the end of each one of the chapters so the idea here is not to summarise every conclusion discussed before but rather to go over some of the most important lessons learned from the development of this project.

**Web 2.0 is not a fad but a real development of World-Wide Web:** A couple of years ago in the case studies module of our masters we had an interesting discussion about the Web 2.0 trying to determine if it was really an advancement in the

World-Wide Web or if it was a market ploy created only for economical reasons. Two years later when this research project was conducted and the literature was reviewed one conclusion is that even if there is not a breakthrough technological development completely associated with the Web 2.0 phenomenon, the importance of the social networks, the ability to collaborate, the user created content and the constant updates by the web users have shifted the mindset of the web-sphere and most of the popular web sites today have incorporated in their platforms an important amount of social components and the user interaction with the content has become the norm. The traditional news media offer blogging platforms on their websites and give the users the ability to comment the news or even to download a podcast, most of the people have heard about Facebook and have their friends and colleges updated using their status, twitter is picking up and the user base have grow and the people is getting into the habit of the microblogging. Even traditional web services like Google offers now the ability to comment the search results, store and share with friends a colleges RSS feeds and create user-generated maps in the maps platform. The World-Wide Web is far away from the model of serving static pages to the visitor. Today the web is social, today we live in the Web 2.0.

**Spam is a threat to the popular Web 2.0 services:** E-mail was one of the first and probably the most service to be abused the unsolicited messages, but in the review performed in this dissertation one of the conclusion that can established is the spam phenomenon can have a negative impact in any service that offers the user interaction. In the literature review it was noticed that the splogs and the fake accounts created in social bookmarking services by the spam users can affect the usefulness and the overall quality of the service. The other important conclusion from this topic is the popularity plays an important role in the amount of spam a service receive. Grays 2.0, this medical social bookmarking system did

not suffer any single spam attack. One of the reasons might be the users needed to authenticate before posting information and the registration and authorization procedures were handled by Google Accounts, but still one of the hypothesis is the service was still begining to ramp up and was not ranking yet in the web search engines. So the popularity might be a factor directly related with the spam.

**Iterative and agile methodologies are suitable for Web 2.0 projects:** Agile methodologies were the working software is the most important part the performance evaluation, the teams regard their customers as the most important stakeholders and were the software development cycles are shorter having a mentality of release soon, release often is in the mindset of the team, are a good methodology for Web2.0 projects. The World-Wide Web is very dynamic and changes very quickly so sequential models such as the waterfall ones are not suitable for the web 2.0 applications.

**Django over the Google App Engine to host starter Web 2.0 applications:** During the development of this application, the exercise of installing and configure the Django Web Framework and the Google App Engine Software Development Kit was performed and it was possible to create a development environment ready to develop, test and deploy a web application in the Google App Engine hosting platform. When the author was reading the documentation about the App Engine he found Django was supported natively and that Google allocated a free quota for hosting the application without any cost, so it was a very attractive solution for this project. During the development of this project this fact was verified and the application is running right now on the Google App Engine using the django web framework. If the application becomes very successful and goes over the quota then the owner need to pay based on the resource usage, but for starter application is a good alternative.

**Python is a very powerful programming language:** One of the tasks related to the web development of this project was to learn how to program in Python, since the selected web framework and hosting platform were created for the development on that language. Even if at the beginning it was difficult to get used to the code blocks based on indentation and not curly braces and the strict enforcement of those rules, once those were assimilated in the coding practices, the development experience was a real joy. The data structures of Python are amazing and the combination of lists, dictionaries and tuples, next to the extensive built-in library allow the developer to perform complex tasks in a few lines of code. Even at the end of the project Python was used to process efficiently the feedback data.

**It is possible to write a dissertation with just open source packages:** The last additional conclusion from this project derives from the process itself of writing this dissertation. One conclusion at the end of this project is that it is possible to write a complex dissertation with multiple chapters, several references, tables, figures, screenshots, etc, using only open source packages. This dissertation was written in the same netbook development machine discussed in the hardware requirements with Ubuntu 8.10 installed. The document was typed in Latex using gedit and texmaker and the bibliography was managed with bibtex using kbibtex. Also the graphical elements were edited with GIMP and the screenshots were performed with ksnapshot. Therefore the whole project was done using only open source packages. This fact really brings down the costs of the dissertation project.

## 9.3 Future Work

### 9.3.1 Medical Bookmarking System

During the evaluation period of the Medical Bookmarking system, the general question about the open functionality was very good to discover there are areas of improvement in the application and where were some features that were not envisioned in the design phase that might be interesting to develop in the next iteration. Some of those are:

**Description comment of each bookmark:** In the current system it is only possible to associate the bookmark with the title or tags. One the suggestion from the evaluators was to add also a general description field to each one of the bookmarks, similar to the “notes” functionality *del.icio.us* offers.

**RSS Feeds:** One of the users suggested to offer the RSS functionality for the user bookmarks or tag bookmarks. Using this functionality the users could add a live bookmark option to their personal bookmark list in their browsers (like in Firefox) and then keep synchronized the Grays bookmarks with the off-line ones. They also could subscribe to the bookmarks created by other users or to the bookmarks associated with one particular tag.

**Add other user’s bookmarks in your list:** Some users suggest adding a quick functionality to get the bookmarks added by other users into the personal list. This feature would be easy to implement and it probably will be a time saver for many users.

**Voting system:** An interesting functionality could be to offer the user the ability to vote and push up or put down a particular bookmark from the repository. This

information could be used for sorting the bookmarks when user issues a query in the search box or when someone is browsing the bookmarks by tag.

**User Interface Improvements:** A couple of quotes from the users: *“more graphics so the website will look more friendly, a more user friendly interface would be nice, the look of the site does not attract people”*. Based on those it is clear that the User Interface could be improved to make it more interesting to the visitors. The next iteration should include a revamp in the UI.

### 9.3.2 Other areas

**Different Web 2.0 Applications:** The scope of this project was limited to the Social Bookmarking Services. But there are many other Web 2.0 services and applications that could be implemented using a similar methodology to the one used in this project. It could be interesting to see social networks, blogging services or other folksonomy based applications running on Django over the Google App Engine.

**Spam Study in Content Management Systems:** One of the drawbacks of this project was that it was not possible to study the spam phenomenon on it because the system was not spammed at all. It would be interesting to use one of the ready to use Content Management System packages or blogging platforms and to study the spam on them.

**Web 3.0:** In some newspapers and technology magazine articles there are beginning to appear terms like Web 3.0, as the next generation of Web services. An interesting research topic would be to investigate if there is really such a term and if it is the future of the World-Wide Web, what would entitle.

## 9.4 Personal Remarks

I have been working during the last three years for one of the most important Internet companies in the world and I have been fortunate enough to have the chance to experience first hand how the web is evolving at a really fast pace. The first year I had the opportunity to witness one small presentation from Tim O'Reilly where he discussed the term of Web 2.0 and at that moment I had to confess I haven't heard about this term before. Now three years later the term is everywhere and there are many services related to the Web 2.0 and the social services. I am thrilled I had the chance to develop this dissertation project in that interesting field, combining my two professional backgrounds.

# Appendix A

## Grays2.0 Code Snippets

### A.1 Configuration Files

#### A.1.1 app.yaml

```
application: medical-bookmarks
version: 1
runtime: python
api_version: 1
```

```
handlers:
- url: /static
  static_dir: static

- url: .*
  script: main.py
```

### A.2 App Engine Django Application Handler Files

#### A.2.1 views.py

```
def search_page(request):
    """Handler for showing the page for searching the bookmarks """
    user = users.GetCurrentUser()
    form = SearchForm()
    bookmarks = []
    show_results = False
```



```

if request.GET.has_key('query'):
    show_results = True
    query = request.GET['query'].strip()
    if query:
        # print query
        form = SearchForm({'query' : query})
        bookmarks = Bookmark.all().search(query).order("-created")

    return respond(request, user, 'search', {
        'form': form,
        'query': query,
        'bookmarks': bookmarks,
        'show_results': show_results,
        'show_tags': True,
        'show_user': True
    })

def user_page(request, username):
    """Handler for showing the User Page in the Bookmarking System"""

    user = users.GetCurrentUser()
    try:
        email = username
        #+ "@gmail.com" Add this part to the appspot.com version
        userbookmarks = users.User(email)
    except:
        raise Http404('Requested user not found.')
    bookmarks = Bookmark.gql("WHERE user = :userp ORDER BY created DESC",userp=user)
    return respond(request, user, 'user_page', {
        'username': userbookmarks,
        'bookmarks': bookmarks,
        'show_tags': True
    })

```

### A.2.2 urls.py

```

from django.conf.urls.defaults import *

urlpatterns = patterns(
    '',

```

```
(r'^$', 'views.index2'),
(r'^index2$', 'views.index'),
(r'^user/(.+)/$', 'views.user_page'),
(r'^user/(.+)/([\s]+)/$', 'views.user_tag_page'),
(r'^delete/$', 'views.bookmark_delete'),
(r'^save/$', 'views.bookmark_save_page'),
(r'^search/$', 'views.search_page'),
(r'^tag/([\s]+)/$', 'views.tag_page'),
(r'^tag/$', 'views.tag_cloud_page'),
)
```

### A.2.3 forms.py

```
import re
from django import newforms as forms

# Form for storing the Bookmarks.

class BookmarkSaveForm(forms.Form):
    url = forms.URLField(
        label='URL',
        widget=forms.TextInput(attrs={'size': 64})
    )
    title = forms.CharField(
        label='Title',
        widget=forms.TextInput(attrs={'size': 64})
    )
    tags = forms.CharField(
        label='Tags',
        required=False,
        widget=forms.TextInput(attrs={'size': 64})
    )
    edit = forms.CharField(
        required=False,
        widget=forms.HiddenInput()
    )

# Form for deleting the Bookmarks.
class BookmarkDeleteForm(forms.Form):
    key = forms.CharField(
        required=False,
        widget=forms.HiddenInput()
```

```

    )
    confirm = forms.CharField(
        required=False,
        widget=forms.HiddenInput()
    )

# Form for querying the bookmark datastore.
class SearchForm(forms.Form):
    query = forms.CharField(
        label = '',
        widget=forms.TextInput(attrs={'size': 32})
    )

```

## A.3 App Engine Django Application Template Files

### A.3.1 base.html

```

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title>MedB - Medical Bookmarks</title>
    <link type="text/css" rel="stylesheet" href="/static/styles.css">
    <script type="text/javascript" src="/static/script.js"></script>
    <script type="text/javascript" src="/static/menu.js"></script>

</head>
<body>
<table width="700" border="0" cellpadding="0" cellspacing="0">

{% include "header.html" %}
{% include "menu.html" %}

<table class="white" width="100%" border="0" cellspacing="0"
    cellpadding="0">
    <tr>
        <td width="3%">

{%block body%}{%endblock%}

{% include "footer.html" %}

```

```
</body>
</html>
```

### A.3.2 header.html

```
<!-- Header Row -->
<tr>
  <td colspan="2">
<div class="headtitle"><h1 id="headertitle"><span
  style="color:yellow; font-weight: bold;">
    Medical Bookmarks</h1></div>
</td>
</tr>
<!-- menu row -->
<tr>
  <td colspan="2" bgcolor="#336699">
<table width="100%" border="0" cellspacing="0" cellpadding="0">
  <tr>
    <td class="hmenu"></td>
    <td class="hmenu">&nbsp;</td>
{%if user %}

{% else %}
{%if sign_in%}<td class="hmenu"><a href="/" class="hmenulink">
  <a href="{sign_in}">Sign in</a></td>{%endif%}
{%endif%}
    <td class="hmenu"><a href="/" class="hmenulink">Homepage</a>
      </td>
    {%if user %}
    <td class="hmenu"><a href="/user/{{user.nickname}}/"
      class="hmenulink">
        {{user.nickname}}'s Bookmarks</a></td>
    {% endif %}
    <td class="hmenu"><a href="/save/" class="hmenulink">
      Add New Bookmark</a></td>
    <td class="hmenu"><a href="/kwd/" class="hmenulink">
      Tag Cloud</a></td>
  <td><table width="100%" border="0" cellspacing="0" cellpadding="0">
```

```

        <tr>
            <td><div align="right">
                <form action="/search/" method="get" name="search">
                    {%if query%}
                        <input type="text" name="query" size="8" value="{{query}}" />
                    {% else %}
                        <input type="text" name="query" size="8" />
                    {% endif %}
                        <input type="submit" value="search" />
                </form>
            </div></td>
        </tr>
    </table></td>
</tr>
</table>
</td>
</tr>
<tr>
    <tr>
        <!-- The height of this spot will tell how big the palid
            blue line will be -->
        <td colspan="2" id="dateholder" class="leftmenu">
            <div align="right" class="whitebold">
                Date: {% include "datetime.html" %}
                |
            <!-- Login Info -->
            {%if user %}
                <b>{{user.nickname}} &lt;{{user.email}}&gt;</b>
                {%if is\_admin%} | <a href="/\_ah/admin">Admin</a>{%endif%}
                {%if sign\_out%} | <a href="{{sign\_out}}">Sign out</a>{%endif%}
            {% else %}
                {%if sign\_in%}<a href="{{sign\_in}}">Sign in</a>{%endif%}
            {%endif%}
            <!-- End Login Info -->
        </div>
    </td>
    </tr>

```

### A.3.3 menu.html

```

<tr>
    <!-- The height of this spot will tell how big the palid
        blue line will be -->

```

```

<td colspan="2" id="spotholder" class="leftmenu"></td>
<!-- -->
</tr>
<tr>
    <td width="119" id="menuholder" class="leftmenu">
        
<div class="latestnews">
<p />
<b>Latest Additions</b>.<br />
{% if latestbookmarks %}
    {% for bookmark in latestbookmarks %}
        - <a href="{{ bookmark.link.url }}" target="_blank">
            {{ bookmark.title }}<a/><br />
{% endfor %}
{% endif %}
<p />
</div>
<div class="signbox" id="sigbox">
<table width="100%" border="0" cellpadding="2" cellspacing="0" bgcolor="#6699CC">
    <tr>
        <td>
            <table width="100%" border="0" cellspacing="0" cellpadding="0">
                <tr>
                    <td></td>
<td></td>
                </tr>
                <tr>
                    <td></td>
<td></td>
                </tr>
            </table>
        </td>
        <td class="leftmenuitem">&nbsp;&nbsp;&nbsp;&raquo;</td>
<td class="leftmenuitem">
{%if user %}
    <a href="{{sign\_out}}" class="yellowlink">Sign Out</a></td>
{% else %}
    <a href="{{sign\_in}}" class="yellowlink">Sign In</a></td>
{% endif %}
    </tr>
</table>
</td>
</tr>
</table>

```

```

</td>
<td width="591" align="left" valign="top" class="main">
<table width="100%" border="0" cellspacing="0" cellpadding="0">
  <tr>
    <td width="3%" id="cornerholder" class="leftmenu">
      
    </td>
    <td width="95%" class="main">&nbsp;</td>
  <td width="3%" bgcolor="#FFFFFF"></td>
  <td width="1%" bgcolor="#CCCCCC"></td>
  </tr>
</table>

```

### A.3.4 footer.html

```

    </td>
    <td width="3%" bgcolor="#FFFFFF"></td>
  <td width="1%" bgcolor="#CCCCCC"></td>
  </tr>
</table>
</td>
</tr>

<tr>
  <td colspan="2" class="footer">&copy; 2008 MedB. Developed by <a href="mailto:webda
</td>
</tr>
</table>

```

### A.3.5 main\_page.html

```

{%extends "base.html"%}
{% block title %}Welcome to Django Bookmarks{% endblock %}
{%block body%}
<h2>Welcome to Medical Bookmarks</h2>
{%if user %}

```

```

<p>Welcome {{ user }}!
    Here you can store and share bookmarks!. To see your bookmarks
        <a href="/user/{{ user }}/">click here</a></p>
{% else %}
    <p>Welcome anonymous user!
        You need to <a href="{{sign\_in}}">login</a> with
            your Google Account
            before you can store and share bookmarks.</p>
{% endif %}
<br />
Feel free to click in the tag cloud in order to see the
    popular bookmarks or
    use the search box on top go get bookmarks related to
    your query<br /><br />

{% if tags %}

<table summary="tag cloud" class="bluebox" cellspacing="0">
<tr>
<td class="blueboxBody">
<div id="tag-cloud">
    {% for tag in tags %}
        <a href="/kwd/{{ tag.name }}/"
            class="tag-cloud-{{ tag.weight }}">
            {{ tag.name }}</a>
    {% endfor %}
</div>
</td>
</tr>
</table>

{% endif %}

{%endblock%}

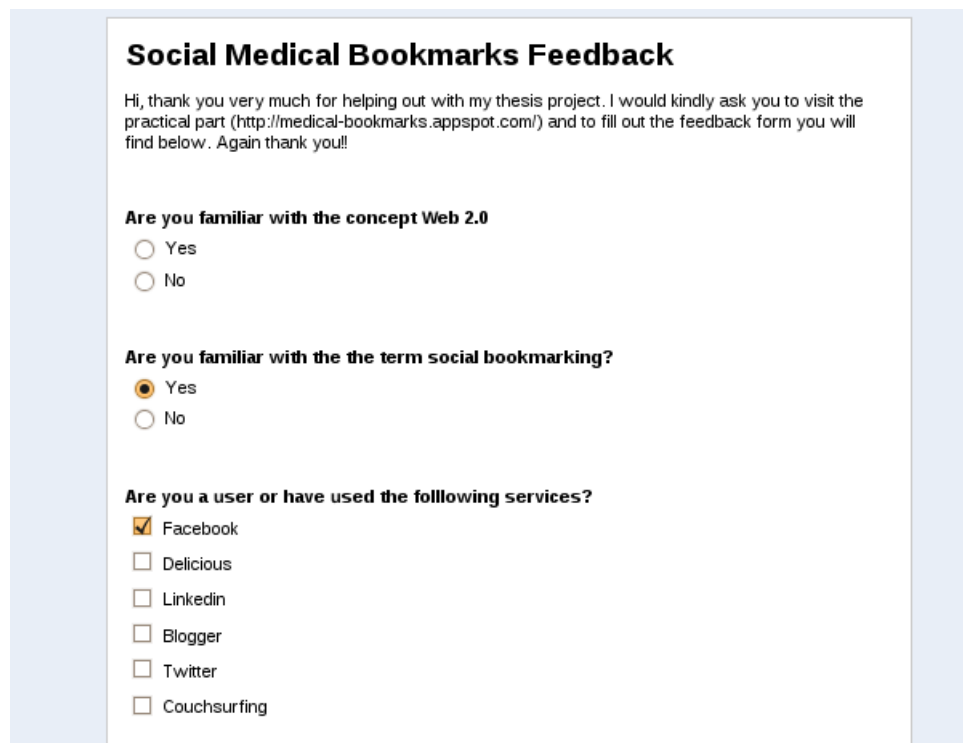
```



# Appendix B

## Evaluation Form

Screenshots from <http://www.tinyurl.com/poll-thesis>



**Social Medical Bookmarks Feedback**

Hi, thank you very much for helping out with my thesis project. I would kindly ask you to visit the practical part (<http://medical-bookmarks.appspot.com/>) and to fill out the feedback form you will find below. Again thank you!!

**Are you familiar with the concept Web 2.0**

☐ Yes  
☐ No

**Are you familiar with the the term social bookmarking?**

☒ Yes  
☐ No

**Are you a user or have used the following services?**

☒ Facebook  
☐ Delicious  
☐ LinkedIn  
☐ Blogger  
☐ Twitter  
☐ Couchsurfing

Figure B.1: Evaluation Form - Part1.

**Did you visit the <http://medical-bookmarks.appspot.com/> website?**

☒ Yes  
☐ No

**Did you browse / search for information in <http://medical-bookmarks.appspot.com/>?**

☐ Yes  
☐ No

**If you answered yes, how easy was to find the information**

1 2 3 4 5

Impossible ☐ ☐ ☐ ☐ ☐ Extremely Easy

**Did you add a bookmark into <http://medical-bookmarks.appspot.com/>**

☒ Yes  
☐ No

**If you answered yes, how easy was to add a new bookmark**

1 2 3 4 5

Impossible ☐ ☐ ☐ ☐ ☐ Very Easy

Figure B.2: Evaluation Form - Part2.

**Do you work in the health care field?**

☒ Yes  
☐ No

**In general how do you rate the site If you rate the site <http://medical-bookmarks.appspot.com/>**

1 2 3 4 5

Awful ☐ ☐ ☐ ☐ ☐ Amazing

**What functionality is missing in the service (and you would like to have)**

Powered by [Google Docs](#)

[Terms of Service](#) - [Additional Terms](#)

Figure B.3: Evaluation Form - Part3.

# Appendix C

## Data Analysis .py script

Python script written for analysing the data from the feedback and generating the graphs (feed.py):

```
import csv

totalrows = 0

yesnoTuple = ('Yes', 'No')
scaleTuple = ('1', '2', '3', '4', '5')

familiar20 = {'Yes': 0, 'No': 0}
familiarSB = {'Yes': 0, 'No': 0}
services = {'Facebook': 0, 'Delicious': 0, 'Linkedin': 0,
            'Blogger': 0, 'Twitter': 0, 'Couchsurfing': 0}
visitGray = {'Yes': 0, 'No': 0}
browseGray = {'Yes': 0, 'No': 0}
easybrowseGray = {'1': 0, '2': 0, '3': 0, '4': 0, '5': 0}
addDBGGray = {'Yes': 0, 'No': 0}
easyaddGray = {'1': 0, '2': 0, '3': 0, '4': 0, '5': 0}
doctor = {'Yes': 0, 'No': 0}
generalscore = {'1': 0, '2': 0, '3': 0, '4': 0, '5': 0}
feedback = []

def analyzer(dictsent, value, params):
    """ It counts the number of times the passed value is
        in the sent parameters and it is assigned to the
        sent dictionary"""
    newvalues = value.split(",")
    for newvalue in newvalues:
        for item in params:
            if newvalue.strip() == item:
```

```

        dictsent[item] = int(dictsent[item]) + 1
    return

def printdic(dictionarysent):
    """It prints the key, value pair of a dictionary
       separated by ; """
    for key in dictionarysent.keys():
        print str(key) + ";" + str(dictionarysent[key])

csvReader = csv.reader(open('feedback.csv'),
                        delimiter=',', quotechar='"')
for row in csvReader:
    totalrows = totalrows + 1
    analyzer (familiar20, row[1], yesnoTuple)
    analyzer (familiarSB, row[2], yesnoTuple)
    analyzer (services, row[3], ('Facebook', 'Delicious',
                                'Linkedin', 'Blogger',
                                'Twitter', 'Couchsurfing'))

    analyzer (visitGray, row[4], yesnoTuple)
    analyzer (browseGray, row[5], yesnoTuple)
    analyzer (easybrowseGray, row[6], scaleTuple)
    analyzer (addDBGray, row[7], yesnoTuple)
    analyzer (easyaddGray, row[8], scaleTuple)
    analyzer (doctor, row[9], yesnoTuple)
    if len(row) >= 11:
        analyzer (generalscore, row[10], scaleTuple)
    if len(row) >= 12:
        feedback.append(row[11])

print "Familiar Web 2.0"
printdic (familiar20)
print "Familiar Social Bookmarking"
printdic (familiarSB)
print "Social Services"
printdic (services)
print "Visited Gray"
printdic (visitGray)
print "Browse Gray"
printdic (browseGray)
printdic (easybrowseGray)
print "Add Gray"
printdic (addDBGray)
printdic (easyaddGray)
print "Doctor"
printdic (doctor)
print "General Score"
printdic (generalscore)

```

```
print "General Feedback"
for item in feedback:
    print item
print "totalrows " + str(totalrows)
```

# References

- Aldhous, P. (2008). How the myspace mindset can boost medical science. *New Scientist magazine*, (2656), 26–27.
- Alpert, J., & Haja, N. (2008). We knew the web was big..  
URL <http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html>
- Angel, E. (2010). *Interactive Computer Graphics : A Top-Down Approach with OpenGL*. Boston, MA: Addison Wesley Longman.
- Augar, N., Raitman, R., & Zhous, W. (2004). Teaching and learning online with wikis.  
URL <http://ascilite.org.au/conferences/perth04/procs/augar.html>
- Barnes, J. (1954). Class and committees in a norwegian island parish. *Human Relations*, 7(1), 39–58.
- Barsky, E. (2006). Introducing web 2.0: Rss trends for health librarians. *J Can Health Lib Assoc*, (27), 7–8.
- Barsky, E., & Purdon, M. (2006). Introducing web 2.0: social networking and social bookmarking for health librarians. *J Can Health Lib Assoc*, (27), 65–67.
- Bates, D., & Gawande, A. (2003). Improving safety with information technology. *New England Journal of Medicine*, (348), 2526–2534.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2006). Manifesto for agile software development.

URL <http://www.agilemanifesto.org/>

Berners-Lee, T. (2008). The website of the world's first-ever web server.

URL <http://info.cern.ch/>

Berners-Lee, T., Fielding, R., & Frystyk, H. (1996). Hypertext transfer protocol – http/1.0.

URL <http://www.ietf.org/rfc/rfc1945.txt>

Blood, R. (2000). weblogs: a history and perspective.

URL [http://www.rebeccablood.net/essays/weblog\\_history.html](http://www.rebeccablood.net/essays/weblog_history.html)

Boulos, M. K., Maramba, I., & Wheeler, S. (2006). Wikis, blogs and podcasts: a new generation of web-based tools for virtual collaborative clinical practice and education. *BMC Medical Education*, (6), 41.

Boyd, D., & Ellison, N. (2007). Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 11(1), 39–58.

Brain, M. (2003). How spam works.

URL <http://computer.howstuffworks.com/spam.htm>

Brain, M. (2005). How wikis work.

URL <http://computer.howstuffworks.com/wiki.htm>

Cadenhead, R., & Smith, G. (2006). The application/rss+xml media type.

URL <http://www.rssboard.org/rss-mime-type-application.txt>

- Castillo, C., Donato, D., Becchetti, L., Boldi, P., Santini, M., & Vigna, S. (2006). A reference collection for web spam. *SIGIR Forum*, 40(2), 11–24.
- Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., & Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2).
- Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 377–387.
- comscore (2007). Bebo becomes the most visited social networking site in the uk.  
URL <http://www.comscore.com/press/release.asp?press=1571>
- Daly, L. (2007). *Next-generation Web Frameworks in Python*. O'Reilly.
- Darling, D. (2006). The ipod ecosystem.  
URL <http://www.nytimes.com/2006/02/03/technology/03ipod.htm>
- delicious.com (2007). Delicious faq.  
URL <http://www.delicious.com/deliciousfaq.html>
- Dimov, V. (2006). Top 5 medical podcasts i listen to.  
URL <http://casesblog.blogspot.com/2006/08/top-5-medical-podcasts-i-listen-to.html>
- djangoproject.com (2008). The django project. Last accessed: Apr 30, 2009.  
URL <http://www.djangoproject.com/>
- dmoz.org (2009). Dmoz social networking.  
URL [http://www.dmoz.org/Computers/Internet/On\\_the\\_Web/Online\\_Communities/Social\\_Networking/](http://www.dmoz.org/Computers/Internet/On_the_Web/Online_Communities/Social_Networking/)
- ec.wmich.edu (2009). List of evaluation glossary terms.  
URL <http://ec.wmich.edu/glossary/glossaryList.htm>



Fowler, M. (2005). The new methodology.

URL <http://martinfowler.com/articles/newMethodology.html>

Fowler, M. (2006). Writing the agile manifesto.

URL <http://www.martinfowler.com/articles/agileStory.html>

García-Molina, H., & Gyöngyi, Z. (May 2005). Web spam taxonomy. *Proceedings of the 1st International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*.

Garrett, J. (2005). Ajax a new approach to web applications.

URL <http://www.adaptivepath.com/publications/essays/archives/000385.php>

Gedda, R. (2009). New twitter app helps with job search.

URL [http://www.pcworld.com/businesscenter/article/163391/new\\_twitter\\_app\\_helps\\_with\\_job\\_search.html](http://www.pcworld.com/businesscenter/article/163391/new_twitter_app_helps_with_job_search.html)

Giles, J. (2005). Internet encyclopaedias go head to head. *Nature*, 438(7070), 900–1.

Giustini, D. (2006). How web 2.0 is changing medicine. *British Medical Journal*, (333), 1283–1284.

Glasner, J. (2001). A brief history of spam, and spam.

URL <http://www.wired.com/techbiz/media/news/2001/05/44111>

Google.com (2008a). Google app engine - the model class.

URL <http://code.google.com/appengine/docs/python/datastore/modelclass.html>

Google.com (2008b). Google app engine - types and properties.

- URL <http://code.google.com/appengine/docs/python/datastore/typesandpropertyclasses.html>
- Google.com (2008c). Google app engine guide.  
URL <http://code.google.com/appengine/docs/>
- Google.com (2008d). Introducing google app engine + blog.  
URL <http://googleappengine.blogspot.com/2008/04/introducing-google-app-engine-our-new.html>
- Gross, R., & Acquisti, A. (2005). Information revelation and privacy in online social networks (the facebook case). In *ACM Workshop on Privacy in the Electronic Society (WPES)*, (pp. 71–80). Alexandria.  
URL <http://www.heinz.cmu.edu/~acquisti/papers/privacy-facebook-gross-acquisti.pdf>
- Halvey, M. J., & Keane, M. T. (2007). An assessment of tag presentation techniques. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, (pp. 1313–1314). New York, NY, USA: ACM Press.
- Hammond, T., Hannay, T., Lund, B., & Scott, J. (2005). Social bookmarking tools (i): A general review. *D-Lib Magazine*, 11.
- Heymann, P., Koutrika, G., & Garcia-Molina, H. (2007). Fighting spam on social websites: A survey of approaches and future challenges. *IEEE Internet Computing*.
- Holovaty, A., & Kaplan-Moss, J. (2007). *The Definitive Guide to Django: Web Development Done Right*. apress.
- Horton, W. K. (2000). *Designing Web-Based Training: How to Teach Anyone Anything Anywhere Anytime*. John Wiley and Sons.

- Hotho, A., Jaschke, R., Schmitz, C., & Stumme, G. (2006a). Information retrieval in folksonomies: Search and ranking. *Lecture Notes in Computer Science*.
- Hotho, A., Jaschke, R., Schmitz, C., & Stumme, G. (2006b). Bibsonomy: A social bookmark and publication sharing system. In *Proceedings of the Conceptual Structures Tool Interoperability Workshop at the 14th International Conference on Conceptual Structures*.
- Hourieh, A. (2008). *Learning Website Development with Django*. Packt Publishing.
- IEEE (1998). Ieee standard for software test documentation. Tech. rep.  
URL [http://ieeexplore.ieee.org/xpls/abs/\\_all.jsp?arnumber=741968](http://ieeexplore.ieee.org/xpls/abs/_all.jsp?arnumber=741968)
- ipswitch.com (2006). Ipswitch, inc., warns that spam continues to rise.  
URL <http://tinyurl.com/ipswitch>
- ISO/IEC Standard (2003). Software engineering – product quality – part 1: Quality model. ISO Standard 9126-1, ISO/IEC.
- Jacobs, I. (2004). Architecture of the world wide web, volume one.  
URL <http://www.w3.org/TR/webarch/>
- Java, A., Song, X., Finin, T., & Tseng, B. (2007). Why we ptwitter: Understanding microblogging usage and communities. *Procedings of the Joint 9th WEBKDD and 1st SNA-KDD*.
- Kan, S. H. (2003). *Metrics and Models in Software Quality Engineering*. Addison-Wesley.
- Kohler, D. (2008). Running django on google app engine.  
URL <http://code.google.com/appengine/articles/django.html>

- Kolari, P., Finin, T., & Joshi, A. (2006a). Svms for the blogosphere: Blog identification and splog detection. *Proceedings of the AAAI Spring Symposium on Computational Approaches to Analysing Weblogs.*
- Kolari, P., Java, A., Finin, T., Mayfield, J., Joshi, A., & Martineau, J. (2006b). Blog track open task - spam blog classification. *Technical report TREC 2006 Blog Track.*
- Koutrika, G., Adjie, F., Gyöngyi, Z., Heymann, P., & Garcia-Molina, H. (May 2007). Combating spam in tagged systems. *Third International Workshop on Adversarial Information Retrieval on the We.*
- Krasner, G. E., & Pope, S. T. (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *ParcPlace Systems, Inc.*
- Kruchten, P. (2001). What is the rational unified process.  
URL [http://www.therationaledge.com/content/jan\\_01/f\\_rup\\_pk.html](http://www.therationaledge.com/content/jan_01/f_rup_pk.html)
- Kruchten, P. (2003). *The rational unified process: an introduction*. Addison-Wesley, 3 ed.
- Luo, J. S. (2007). Social networking: Now professionally ready. *Primary Psychiatry*, 14(2), 21–24.
- Marlow, C. (2004). Audience, structure and authority in the weblog community. *International Communication Association Conference.*
- Mathes, A. (2004). Folksonomies - cooperative classification and communication through shared metadata. *Computer Mediated Communication.*
- Mcafee ICF International (2009). The carbon footprint of email spam report. Tech. rep.  
URL [http://img.en25.com/Web/McAfee/CarbonFootprint\\_12pg\\_web\\_REV\\_NA.pdf](http://img.en25.com/Web/McAfee/CarbonFootprint_12pg_web_REV_NA.pdf)

McConnell, S. (1996). *Rapid Development*. Microsoft Press.

McLean, R., Richards, B., & Wardman, J. (2007). The effect of web 2.0 on the future of medical practice and education: Darwinian evolution or folksonomic revolution. *Medical Journal of Australia*.

McLellan, D. (2005). Very dynamic web interfaces.

URL <http://www.xml.com/pub/a/2005/02/09/xml-http-request.html>

medworm.com (2009). Medical blog tag cloud.

URL <http://www.medworm.com/rss/blogtags.php>

Mind Tree Blog (2008). Most popular web application frameworks.

URL <http://www.hurricanesoftwares.com/most-popular-web-application-frameworks>

Molina, S. T., & Borkovec, T. D. (1994). The Penn State worry questionnaire: Psychometric properties and associated characteristics. In G. C. L. Davey, & F. Tallis (Eds.) *Worrying: Perspectives on theory, assessment and treatment*, (pp. 265–283). New York: Wiley.

NCSA University of Illinois (2001). Common gateway interface.

URL <http://hoohoo.ncsa.illinois.edu/cgi/intro.html>

nexen.net (2008). Php statistics for june 2008.

URL [http://www.nexen.net/chiffres\\_cles/phpversion/18519-php\\_statistics\\_for\\_june\\_2008.php](http://www.nexen.net/chiffres_cles/phpversion/18519-php_statistics_for_june_2008.php)

Ofcomp (2008). Social networking, a quantitative and qualitative research report into attitudes, behaviours and use.

URL [http://www.ofcom.org.uk/advice/media\\_literacy/medlitpub/medlitpubrss/socialnetworking/report.pdf](http://www.ofcom.org.uk/advice/media_literacy/medlitpub/medlitpubrss/socialnetworking/report.pdf)

O'Reilly, T. (2005). What is web 2.0.

URL <http://www.oreillynet.com/lpt/a/6228>

O'Reilly, T. (2006). Web 2.0 compact definition: Trying again.

URL <http://radar.oreilly.com/archives/2006/12/web-20-compact.html>

Oudot, L. (2003). Fighting spammers with honeypots.

URL <http://www.securityfocus.com/infocus/1747>

Papadimoulis, A. (2007). The great pyramid of agile.

URL <http://thedailywtf.com/Articles/The-Great-Pyramid-of-Agile.aspx>

Paulson, L. (2005). Building rich web applications with ajax. *IEEE Computer*, 38(10), 14–17.

Perkins, A. (2001). The classification of search engine spam.

URL <http://www.silverdisc.co.uk/articles/spam-classification/>

Pes, C. (2009). Metodologia de la programacion.

URL [http://www.carlospes.com/curso\\_de\\_ingenieria\\_del\\_software/01\\_02\\_metodologia\\_de\\_la\\_programacion.php](http://www.carlospes.com/curso_de_ingenieria_del_software/01_02_metodologia_de_la_programacion.php)

postini.com (2008). Postini stattrack.

URL <http://tinyurl.com/634m8j>

Rahkila, M. (2006). *Agent-based Method for Self-study Interactive Web-based Education*. Ph.D. thesis, Helsinki University of Technology.

Reardon, M. (2009). At&t uses twitter during service outage.

URL [http://www.pcworld.com/businesscenter/article/163391/new\\_twitter\\_app\\_helps\\_with\\_job\\_search.html](http://www.pcworld.com/businesscenter/article/163391/new_twitter_app_helps_with_job_search.html)

Reed College (2007). Latex your document.

URL <http://web.reed.edu/cis/help/LaTeX/index.html>

Risley, C. (2008). Glass box testing.

URL <http://www.cse.fau.edu/~maria/COURSES/CEN4010-SE/C13/glass.htm>

Royce, W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON*, 26(1), 9.

RSS Advisory Board (2009). Rss 2.0 specification.

URL <http://www.rssboard.org/rss-specification>

Sanderson, D. (2008). *Programming Google App Engine: Rough Cuts Version*. O'Reilly.

Schools, W. (2009). Ajax tutorial.

URL <http://www.w3schools.com/ajax/default.asp>

Sethi, S. K. (2009). Micro-blogging, latest tool in the web 2.0. *Indian Pediatrics*, (46), 188.

Smashing Magazine (2007). Tag clouds gallery: Examples and good practices.

URL <http://www.smashingmagazine.com/2007/11/07/tag-clouds-gallery-examples-and-good-practices/>

Smith, G. (2004). Atomiq: Folksonomy: social classification.

URL <http://atomiq.org/archives/2004/08/folksonomysocialclassification.html>

spamhaus.org (2008). The definition of spam.

URL <http://www.spamhaus.org/definition.html>

Sterman, J. (2006). Learning from evidence in a complex world. *American Journal of Public Health*, (96), 505–514.

- Surowiecki, J. (2004). *The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business, economies, societies, and nations*. Doubleday Books.
- Templeton, B. (2001). Origin of the term spam to mean net abuse.  
URL <http://www.templetons.com/brad/spamterm.html>
- Varlamis, I., & Apostolakis, I. (2007). Medical informatics in the web 2.0 era.  
URL <http://wim.aueb.gr/iraklis/Varlamis-papers/C22.pdf>
- Westland, J. C. (2002). The cost of errors in software development: evidence from industry. *Journal of Systems and Software*, 62(1), 1–9.
- Wetzker, R., Zimmermann, C., & Bauckhage, C. (2008). Analyzing social bookmarking systems: A del.icio.us cookbook. In *Mining Social Data (MSoDa) Workshop Proceedings*, (pp. 26–30). ECAI 2008.  
URL [http://robertwetzker.com/wp-content/uploads/2008/06/wetzker\\_delicious\\_ecai2008\\_final.pdf](http://robertwetzker.com/wp-content/uploads/2008/06/wetzker_delicious_ecai2008_final.pdf)
- Wong, E. (1999). *Artistic Rendering of Portrait Photographs*. Master's thesis, Cornell University.
- Wu, B. (2007). *Finding and Fighting Search Engine Spam*. Ph.d. dissertation, Lehigh University.