Acknowledgements

First and foremost, I would like to thank Professor Hans-Wolfgang Loidl, who supervised me during this project and was always eager to provide help and suggestions of how to improve the system.

I would also like to thank Amir Ghaffari who developed a nice graphical user interface for the application that is the focus of this project.

Last but not least, I would like to thank everybody who is committing code to open source projects: without your help it would not have been possible to create this thesis or the application the thesis is based on.
Abstract

Taking into the account the increasing numbers of research articles that are sent to being peer reviewed every year it is no surprise that the current system of publishing and peer review is not capable of handling them: the argument of it being too slow is the one heard most often.

However, in literature new systems have often been proposed to help deal with this problem - but most of them were never realised. And those that were, seldom provided features that would ease the burden on reviewers and increase the speed of peer reviewing.

As a consequence this paper will propose a system to unify the approaches of e-Publishing and Web 2.0. It will attempt to develop a new system that allows everyone to rate a publication, but that still allows academics to perform a peer review. Moreover, the peer review process should be simplified by using intelligent recommendations thanks to the harnessing of the collective intelligence - a core concept of the Web 2.0 canon.

Furthermore, the system shall introduce the ways of knowledge management and dissemination that are already common in well known Web 2.0 applications, like tags.
CONTENTS

Contents

List of Figures

List of Tables

Listings

1 Introduction

2 Literature Review

3 Requirements Analysis

4 Design & software architecture

5 Implementation

III

VII

VII

IX

1

4

23

23

27

28

31

31

32

34

37

45

57

57

60

65
# ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Active Server Pages</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>BNF</td>
<td>Backus-Naur-Form</td>
</tr>
<tr>
<td>CSRF</td>
<td>Cross-site request forgery</td>
</tr>
<tr>
<td>DAC</td>
<td>Discretionary Access Control</td>
</tr>
<tr>
<td>ERM</td>
<td>Entity-relationship model</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Description Language</td>
</tr>
<tr>
<td>JSON</td>
<td>Javascript Object Notation</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>MAC</td>
<td>Mandatory Access Control</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>OAI</td>
<td>Open Archives Initiative</td>
</tr>
<tr>
<td>OJS</td>
<td>Open Journal System</td>
</tr>
<tr>
<td>OOP</td>
<td>Object Oriented Programming</td>
</tr>
<tr>
<td>ORM</td>
<td>Object Relational Mapper</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Source Initiative</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PMH</td>
<td>Protocol for Metadata Harvesting</td>
</tr>
</tbody>
</table>
CONTENTS

PHP  PHP: Hypertext Preprocessor
RBAC  Role-based Access Control
REST  Representational State Transfer
RIA   Rich Internet Application
RSS   Real Simple Syndication
SFP   Subject Focal Point
SHA1  Secure Hash Algorithm 1
SOAP  Simple Object Access Protocol
SQL   Structured Query Language
SUS   System Usability Scale
UK    United Kingdom
URI   Uniform Resource Identifier
URL   Uniform Resource Locator
WSDL  Web Service Description Language
XML   Extensible Markup Language
XSS   Cross-site scripting
LIST OF FIGURES

2.1 Peer review process flow chart. ........................................... 10
2.2 Basic Role-based Access Control (RBAC) model (modified from Sandhu et al. (1996, p. 41)). ........................................... 19
4.1 Utilized technologies ......................................................... 34
4.2 Decomposition into subsystems. ........................................... 36
4.3 Data-model for the publication management system. .............. 44
4.4 View package (with an excerpt of all resources). ....................... 54
4.5 Insertion module. ............................................................ 55
5.1 Model-View-Controller (MVC) in django. ................................ 58
6.1 Participant’s degrees ........................................................ 68
6.2 Performance of system for GET requests. .............................. 71
6.3 Performance of system for BibTeX POST-requests. ................. 72
8.2 Participant’s computer knowledge ....................................... 84
8.3 Participant’s administration skills ........................................ 84
8.4 Participant’s on online rating/commenting/peer reviewing .......... 86
8.5 Histogramm of the System Usability Scale (SUS) data. ............ 86
8.6 Front page of the administration interface. ............................ 87
8.7 Admin interface to add a publication. .................................. 88
8.8 Admin interface to modify a user. ....................................... 89
8.9 Survey (page 1) .............................................................. 108
8.10 Survey (page 2) ............................................................ 109
8.11 Survey (page 3) ............................................................ 110

LIST OF TABLES

2.1 Comparison of e-Publishing tools. ...................................... 21
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>SUS score calculation</td>
<td>69</td>
</tr>
<tr>
<td>8.1</td>
<td>BibTeX data fields (Part 1)</td>
<td>81</td>
</tr>
<tr>
<td>8.2</td>
<td>BibTeX data fields (Part 2)</td>
<td>82</td>
</tr>
<tr>
<td>8.3</td>
<td>BibTeX data fields (Part 3)</td>
<td>83</td>
</tr>
</tbody>
</table>
LISTINGS

5.1 Calling a template in the view. .......................... 58
5.2 Factory method to obtain an inserter .................. 61
5.3 Building a generic search in Django .................. 62
5.4 Structured Query Language (SQL)-injection vulnerability ..... 65
8.1 gfx/ws/core_web_service/templates/xml/author.xml .... 90
8.2 gfx/ws/core_web_service/templates/xml/authors.xml ... 90
8.3 gfx/ws/core_web_service/templates/xml/comment.xml .... 90
8.4 gfx/ws/core_web_service/templates/xml/comments.xml ... 91
8.5 gfx/ws/core_web_service/templates/xml/esteem.xml .... 92
8.6 gfx/ws/core_web_service/templates/xml/keyword.xml .... 92
8.7 gfx/ws/core_web_service/templates/xml/overview.xml ... 92
8.8 gfx/ws/core_web_service/templates/xml/papergroup.xml ... 93
8.9 gfx/ws/core_web_service/templates/xml/papergroups.xml .. 94
8.10 gfx/ws/core_web_service/templates/xml/peerreview.xml ... 95
8.11 gfx/ws/core_web_service/templates/xml/peerreviews.xml ... 95
8.12 gfx/ws/core_web_service/templates/xml/peerreviewtemplate.xml 96
8.13 gfx/ws/core_web_service/templates/xml/peerreviewtemplates.xml 96
8.14 gfx/ws/core_web_service/templates/xml/publication.xml ... 96
8.15 gfx/ws/core_web_service/templates/xml/publications.xml ... 101
8.16 gfx/ws/core_web_service/templates/xml/rating.xml .... 101
8.17 gfx/ws/core_web_service/templates/xml/ratings.xml ...... 102
8.18 gfx/ws/core_web_service/templates/xml/referencematerial.xml . 102
8.19 gfx/ws/core_web_service/templates/xml/referencematerials.xml 102
8.20 gfx/ws/core_web_service/templates/xml/researcharea.xml ... 103
8.21 gfx/ws/core_web_service/templates/xml/researchareas.xml ... 103
8.22 gfx/ws/core_web_service/templates/xml/tag.xml .......... 103
8.23 gfx/ws/core_web_service/templates/xml/tags.xml ........ 104
8.24 gfx/ws/core_web_service/templates/xml/user.xml ........ 104
8.25 gfx/ws/core_web_service/templates/xml/users.xml ........ 105
8.26 gfx/ws/core_web_service/templates/xml/vote.xml ....... 105
8.27 BibTeX parser Backus-Naur-Form (BNF) ............... 106

IX
INTRODUCTION

“If I have seen further it is only by standing on the shoulders of giants.”
—Sir Isaac Newton

This quote from Sir Isaac Newton sums up the main purpose of scholar activities: utilise the work of other to conduct new research.

But to be able to rely upon the work of others, their research needs to be correct - one way of checking the quality of papers is peer review, a process that has been in place for over 350 years.

However, in recent times the system of peer review and journal publishing has received more and more critique, as it is no longer able to cope with the staggering amount of research that is conducted: the established system can not process the nearly 2.5 million articles that are produced every year (Hull et al. 2008, p. 1).

But there seems to be a solution: utilising the methods that were popularized by the Web 2.0 boom might lessen the burden on reviewers and still allow the academic system to keep on verifying the quality of research.

Therefore this paper will lay the basis for a core platform for knowledge management and publishing that utilises Web 2.0 techniques, but still remains faithful to the original way of quality assessment in science: the peer review process.

The main goals of this project will be to:

**Develop a knowledge management platform:** A system will be provided to allow users to upload their documents to the system.

Apart from allowing only articles to be uploaded, the system will also support new venues for knowledge dissemination and management by supporting newer publication types like blogs. These new publication types can be subject to the same quality assessment as normal articles.

**Allow quality assessment of entries:** As part of the system users will be able to rate, as well as comment on, the uploaded documents informally.

Moreover, it shall be possible to initiate and perform a peer review process for a document: this requires the system to establish two user bases - an
“anonymous” one¹ that can only rate the entries and a verified academic one that can take part in a peer review. This system was referred to as a system of “two different kinds of peers” by Anderson (2006b) while evaluating an open peer review process for the journal Nature.

Extending Anderson’s (2006b) idea, the developed system will provide an esteem based reputation system. This will prove useful in assessing the reliability of comments that were made by the users of the system: the more (useful) comments a user provides the higher his rating will become: the usefulness can be voted upon by other users.

Utilize Web 2.0 technologies: Users will be able to improve the system by using its Web 2.0 features like rating of and commenting on publications as well as the ability to tag entries. By utilizing this user generated content the system can then propose interesting articles based on tags or ratings.

The system utilizes these Web 2.0 technologies as a post-publication filter. Moreover, the Web 2.0 feature set will allow authors and readers to maintain a bi-directional communication relationship: by adding comments a user may notify the author of a document about problems in the recent version that might be fixed when a new version is uploaded.

Utilize data from other systems: The system will leverage data from other systems to provide further metadata for a publication. This data can, for example, be fetched from CiteSeerX².

This feature set will be developed in two different projects: a second project will develop a Rich Internet Application (RIA) that will make the features more accessible to end users, whereas this project will focus on the following issues:

Database design: A main focus will be to design an extensible database to allow the storage of publication and author information that facilitates the use of Web 2.0 technologies to access and enhance the data.

¹These users will only be identifiable via a user name that will not be tied to real world information.
²http://citeseer.ist.psu.edu/
Interface design: To allow the usage of the system from the RIA (and possibly other systems as well) this project will design an interface to the database that can be consumed by other applications. This includes the design of data representations that will be returned by the service to other systems.

User management: The system will also focus on the management of the users that will use the system, as well as providing an interface for administrators to manage the system.

Interface to external sites: To use existing knowledge that is already present in the Web, the system will also focus on the integration of external sites like CiteSeerX and provide a bridge to those.

The necessary background information for each of these different concepts will be discussed in the literature review in chapter 2, followed by a detailed explanation of the requirements in chapter 3.

The following chapter 4 will provide an in-depth explanation of the design of the system, where as chapter 5 will highlight certain implementation specific aspects of the system.

The chapter Evaluation will evaluate the developed system to verify its suitability according to the requirements.

The last chapter will provide a conclusion of the development of the system.
LITERATURE REVIEW

This chapter will provide an overview over the different subareas that will play a part in the development of the system. Definitions, explanations and the different positions found in the literature concerning these topics will be the central part of this chapter. When appropriate it will also be noted how the new system that shall be implemented will make use of these techniques.

2.1 Web 2.0

The term Web 2.0 was first used by Dale Dougherty (the vice-president of O’Reilly Media) in a brainstorming session with MediaLive International (O’Reilly 2007, p. 17). However, he did not provide a definition for the term. A first definition was provided by Tim O’Reilly in his paper “What is Web 2.0: Design patterns and business models for the next generation of software” (O’Reilly 2007). In this paper he describes several ideas and principles that, taken together, define the Web 2.0 as O’Reilly and Dougherty see it:

Web as platform: The web is simply a platform to build new services on, it acts as channel for these services.

Harnessing the long tail: Compared to older Web 1.0 applications that tried to appeal solely to the mass market, newer services tend to reach the entire web. These new services take into account the long tail of customers that were ignored in the former era.

Harnessing collective intelligence: This principle refers to the fact that a group of people can make better decisions than even the most capable individual. This fact is also known as the Wisdom of Crowds (see Surowiecki and Silverman (2007) for more information).

This principle is often associated with Metcalfe’s law about network effects: the law claims that the value of a network is proportional to the square of its connected clients (however, recent literature claims the value is equal to $n \times \log(n)$ the number of connected clients (Andersen 2007, p. 4)): the more
people are using a service, the better the service gets. This is also known as
of the architecture of participation.

O’Reilly argues that this collective intelligence can be used to create a filter
for interesting information.

Data on a epic scale: One of the core features of the Web 2.0 is the fact that the
magnitude of data assembled by the big corporations is accessible by ev-
everyone and can be used to implement new functionality on top of it, thus
creating new services. This process is also referred to as “mash-up”.

End of software release cycle: Compared to desktop applications, the services
offered in the new Web should be developed in a state of a “perpetual open
beta” with new features being deployed to the live system. Moreover the
user should be able to actively participate in the development: for example
by deciding on the new features.

Lightweight programming models: The new web should be based on lightweight
services and loosely coupled systems that allow easy syndication of data and
are designed for “hackability”: developers should be able to utilize the al-
ready existing service to add value to it.

Software above the level of a single device: In 2005 O’Reilly already proclaimed
that Web 2.0 is about targeting more platforms than just the Personal Com-
puter (PC). This is more important than ever before, taking into account the
increasing numbers of tablets and smartphones that are in use.

Generally these ideas are agreed upon in the literature, the main difference is
the importance attributed to each of the principles.

Some authors, like Levy (2009, pp. 121-122) or Andersen (2007, p. 14) tend
to focus more on the principle of user generated content, explaining the differ-
ent kinds of users (passive, minimal active and collaborative active) as well as the
reasons why users will participate in the production of content: the drive to par-
ticipate in the Web 2.0 is based on the idea to gain reputation in a culture where
“getting noticed is everything” (Anderson 2006a, p. 74).
However, there are a few authors that do not see any innovation in the Web 2.0, but see it simply as the realisation of the ideas that Tim Berners-Lee already envisioned for the Web 1.0: the web as a "single, global information space" that “everyone would be able to edit” (Andersen 2007, p. 5).

For the remainder of this work it is of no importance if the concept of the Web 2.0 is actually the realisation of the ideas of the Web 1.0 or not. The important part that needs to be taken into account for the development of the new e-publishing platform are the principles described in this chapter.

2.2 Knowledge management

Before the concrete subject areas of knowledge management and knowledge dissemination can be reviewed a short section will define the different kinds of knowledge.

2.2.1 Knowledge

The term knowledge has many different definitions in contemporary literature. This section will not explore all definitions, but highlight the most common and - for this work - relevant ones.

One of the frequently used ones, defines knowledge by contrasting it with information and data (Stelzer 2009; Groff and Jones 2003, pp. 2-3; Lucko and Trauner 2003, p. 7):

Data: Data is seen as the raw representation existing without any meaning by itself.

Information: Information is generated when data is given a meaning by adding a context to it.

Knowledge: Knowledge is obtained when information is combined with understanding.

This definition often follows up by explaining the three different forms of knowledge that can exist (Groff and Jones 2003, p. 3; Lucko and Trauner 2003, p. 8):
Explicit knowledge: Describes codified knowledge (knowledge that has been written down).

Implicit knowledge: Knowledge that is bound to a certain person and has not yet been written down.

Tacit knowledge: A subgroup of implicit knowledge that is considered especially hard to write down, as it includes intangible factors like perspective and belief.

Knowledge management that adheres to this definition of knowledge defines its main use in the transformation of implicit and tacit knowledge to explicit knowledge. This way it enables more people to benefit from the knowledge.

However, there are other definitions: Gaines (1993, pp. 9-10) defines knowledge as a justified, true belief and explains two different perspectives of knowledge in respect to academic journals:

Objective perspective: This perspective emphasizes the quality of the product (the scientific article), as well as its linkage to other articles.

Social perspective: This perspective emphasizes the producers and their quality. This is an important facet for the system and will be studied in more detail in section 2.3, when reviewing the peer-review process.

2.2.2 Knowledge management

According to Albena et al. (2009, p. 2), there are two main approaches to knowledge management nowadays: the traditional and the conversational (or interactive) approach. This distinction is supported by Lee and Lan (2007, p. 49).

The traditional approach tries to built a collection of knowledge in a central repository, whereas the conversational approach tries to facilitate knowledge creation through the integration and collaboration of knowledge workers.

This distinctions can also be seen in the two different types of knowledge management applications. According to Zack (1999) these two types are integrative and interactive applications.
Integrative applications focus on the explicit storage of knowledge in a repository. This repository can then be accessed by users of the knowledge management system.

Interactive applications in contrast focus on the exchange of tacit knowledge and try to facilitate knowledge sharing.

The system that will be developed will try to make use of both approaches, allowing publications (or metadata of publications) to be uploaded and, at the same time, allow users to comment, rank and tag these artifacts, thus creating new knowledge.

2.2.3 Knowledge Management and Web 2.0

As can already be seen from the different approaches to knowledge management, the conversational approach tends to focus a lot on user participation - a principle of the Web 2.0 as defined in section 2.1.

However, there are also authors that tend to think that the Web 2.0 is not able to improve and complement knowledge management - an example for this view is Snowden (2007), who proclaims that the Web 2.0 ignores the complexity of people and that knowledge management is too complex to make use of the new Web.

Yet, many other authors tend to believe that Web 2.0 concepts can in fact be used to improve knowledge management: Levy (2009, pp. 129-131) argues that the principles of Web 2.0 and knowledge management resemble each other closely and the main differences can be found in centralization and control strategies employed: knowledge management still tends to be more centralized and controlled compared to the Web 2.0’s decentralised fashion.

Moreover Levy (2009, p. 132) states that “when people are socialising, even in a work context, they are much happier to share their thoughts and their experiences”.

Lee and Lan (2007, p. 50) even list matching objectives between the principles of knowledge management and Web 2.0 including contribution, sharing, collaboration and dynamic content. All of these objectives will be taken into account, when implementing the new system: information can be contributed by users and will be collaborated on, thus allowing changes to the content by users ren
Taking these points into account and comparing them to the principles stated in section 2.1 it seems that Web 2.0 seems very capable of improving the current state of knowledge management.

2.3 Publishing & peer review

This section will start with an introduction of the important concepts of peer review and publishing. This will be followed by a section about the different views authors have about peer review and publishing in the Web 2.0 era.

The topics will be discussed together as they are strongly correlated to each other.

2.3.1 Peer review & publishing - the process & principles

The first journal, and with it the first peer-review process, was established in 1665 by Henry Oldenbourg for the Royal Society of London (Parliament 2004).

The main reason for its introduction was to establish a way to track precedence for discoveries, allowing the discoverer to take the credit for his work. This still remains one of the main tasks of the peer review process today. However, nowadays the peer review process has to fulfill other roles as well:

According to Gaines (1993, p. 13) and Kelty et al. (2008, p. 1000) it is part of the peer review process to “verify truth, justification and correct attribution” and to make sure that published material meets a certain quality standard.

This duty, to ensure the quality of an article, has to be seen critically as it may be misinterpreted by non-scholars: it is not the duty of the peer review process to verify the correctness of a paper, but only to check that a submitted article is not obviously wrong (Ginsparg 2002).

And - similar to the first peer review process - most peer reviews are still tied to the publishing process of a journal.

The process of peer reviewing in publishing can be summarised as follows:

1. A paper is submitted for publication.
2. An editor will pre-screen the paper and decide if it is suitable for the journal (or conference).

3. If suitable the paper will be sent to reviewers.

4. Based on the reviews the paper will either be rejected or accepted.

5. If the reviews tend to be undecided the paper can be resent to additional reviewers.

A simple flow-chart of the process is provided for easier visual reference:

![Peer review process flow chart](image-url)

**Figure 2.1:** Peer review process flow chart.

Depending on the receiver the paper is submitted to, there might be slight variations in the process (Wager et al. 2002, pp. 5-8):
1. In a greater journal an editorial board will perform the pre selection of articles.

2. The same editorial board might decide on the final publication or rejection of a paper.

3. Conference papers will be pre-selected by a conference chair.

4. Conference papers will be reviewed and discussed on a conference - this can alter the outcome of the decision.

There are certain variations to this process that intend to ensure a neutral evaluation of the papers (Fröhlich 2002, p. 3; Dall'Aglio 2006, p. 8):

**Blind refereeing:** Blind refereeing describes a peer review process with different levels of anonymity between the referee, the editor and the author of a paper.

- **Single blind refereeing:** In this scenario the author of the paper does not know the identity of the reviewer, but the reviewer is aware of the author's identity.

- **Double blind refereeing:** Like single blind refereeing, but the referee does not know the identity of the author.

- **Triple blind refereeing:** Like double blind refereeing, but even the editor does not know the identity of the author.

**Open peer review:** In this process the author does know the referee who is reviewing his or her paper.

Taking into account a study performed by Ponte and Simon (2009, p. 11) researchers prefer double blind refereeing to the other models, as they assume it to be the one that best ensures a good quality of results: an important factor that needs to be taken into account when developing a new e-publishing system.

Even though journals provide the infrastructure for the current peer review process, it is not their only function. Van de Sompel et al. (2004) and Dall'Aglio (2006, p. 3) provide an overview over the different roles and functions a journal
publisher - or in general, as Van de Sompel calls it, a “scholarly communication system” needs to offer:

Registration: A scholarly communication system must allow the establishing of claims of precedence.

Certification: It must validate the scholarly claim.

Awareness: It must allow scholars to stay aware of new claims and discoveries.

Archiving: It must record the scholarly findings over time.

Rewarding: It must reward scholars based on their performance in the system.

This point is of great importance taking into account that scholars participating in the system as referees do not get any monetary compensation for their activity and their only reward is an improved reputation.

2.3.2 Publishing, peer review and Web 2.0

Having explained the workings of the current system and the principles it was founded on, this section will describe the state of the current approach and explain some of the changes that have already tried to improve the system.

2.3.2.1 State of the current system

That it has been used for the last 350 years is an argument that is often used to describe the success of the peer review and publishing model currently in place. And there are benefits to the model that are summarised in Ware (2008, p. 12):

Improvement in quality: Even though there exists little evidence that peer review really helps in improving the quality of papers, many academics seem to believe this: according to studies performed by Ponte and Simon (2009, p. 10) and Ware (2008, p. 12) 50 or even 90% of the academics that took part in the surveys believed peer review helped to improve quality.

Filtering output: Peer review acts as a valid tool to filter out papers of low impact. Especially high quality journals will only accept the papers that adhere to
CHAPTER 2. LITERATURE REVIEW

high quality standards. This can be seen as a “seal of approval” for the peer reviewed material. Researchers reading these journals are thus spared the great amount of low-quality research that many complain about (Bauerlein et al. 2010).

However, a great amount of critique concerning peer review can be found as well:

Fröhlich (2002, pp. 4-5) claims that only a small number of publications that were send to a publisher for peer review (and later published) were really peer reviewed.

Moreover he criticises that only a small number of reviewers is really active and, as a consequence, those can enact control over the published papers. This point of critique is supported by claims from Rowland (2002, p. 5).

Fröhlich (2002, p. 5), Dall'Aglio (2006, pp. 7-8) and Ware (2008, pp. 16-17) all conducted surveys, whose findings verify that the peer review process fails to detect obvious faults in papers submitted: only two third of major errors were detected according to Fröhlich (2002, p. 5), whereas Dall'Aglio (2006, p. 8) even claims that only “2 out of 8 defects” were found. Ware's (2008, p. 16) findings provide nearly the same results with the exception of claiming that 16% did not find any errors at all.

Dall'Aglio (2006, p. 7), Rowland (2002, p. 5) and Ware (2008, pp. 16-17) point out further weak points of the peer review system:

1. The possibility of bias: commonly denoted forms of bias include nationality, gender and language bias. The greatest problems were encountered by female academics and non-native English speakers.

2. The abuse of a referee's position to steal ideas and commit plagiarism: this can include the delay of a publication to publish own results first or the complete theft of a publication. However, Rowland (2002, p. 6) states that this is a rather rare problem. Abuse on the author's side can include the utilisation of a technique called “salami publishing”, which refers to the practice of producing too many articles from just one conducted research.
3. *Fraud and misconduct*: compared to stealing an idea, fraud and misconduct refer to the practice of using false data or forging data in the first place.

4. The *delay* and *costs* of the peer review process: an often critiqued point of peer review is the fact that it delays the publication process. The studies conducted by Ware (2008, p. 17) list delays between 80 days and over six months as possible. The cost aspect has often been researched and as a general guideline Donovan's (1998) results of 50£-200£ per article seem reasonable and were mainly verified by Rowland's (2002, pp. 6-9) findings.

However, not only the peer review process is critiqued in the literature, the publishing process and the journal medium seem to have several drawbacks as well:

Gaines (1993, p. 7) already pointed out several problems of the journal:

Each volume contains material that might be uninteresting for many subscribers.

Another point are the long delays between accepting and publishing an article: Gaines claims that those can be between one and two years.

Moreover, the journals consume a big amount of library space. This is the case even though journals are limited in size: a fact often critiqued by authors as it forces editors and publishers to issue strict limits on the page number of an article.

Another often stated problem is the fact that journals deal inadequately with non-textual material (Van de Sompel et al. 2004; Wood 1998, pp. 195-196).

Apart from these points, journals also have to face ethical concerns by authors (Jeffery 2006, p. 10): the fact that they publish research results that were funded by the public sector and demand payment for it, is frowned upon by certain researches (refer to *The Open access movement* on page 16 for more information about this problem).

Moreover many authors fear that their desired *research impact* is not reached when publishing in a journal.

Apart from that, it is common for authors to loose their *copyright* to the publisher of a journal as soon as the article is published. 83,9 % of all authors would
CHAPTER 2. LITERATURE REVIEW

rather retain their copyright or transfer the article to the open domain (Ponte and Simon 2009, p. 15) instead of losing it to the publisher.

However, there are not only disadvantages to be found: Gaines (1993, p. 6) points out certain advantages like an experienced staff and a group of competent referees as great strengths of the old system.

And Smith (1999, p. 81) and Kelty et al. (2008, p. 1005) both conclude that a new system has to satisfy the same needs and should built upon the strengths of the old system (referred to as “maximum bootstrapping”).

2.3.2.2 New models

Some (as of yet only theoretical) new journal models have also been proposed:

The deconstructed journal: This model is based on an idea from Smith (1999). He proposes a journal based around a concept called “Subject Focal Point (SFP)”. These SFPs represent portals that link to relevant articles. The articles are neither hosted nor evaluated by the provider of the journal - the quality checking is deferred to “evaluator organisations” and it is the duty of the author to let his articles be peer reviewed. The main advantage of this model is that it shall prevent information scattering and long publication times.

“The system that scholars deserve”: This system, proposed by Van de Sompel et al. (2004), is based around an Open Archives Initiative (OAI)-compliant service that should automatically select possible reviewers for a paper¹. This selection should be based on an analysis of the bibliography of a paper. Once the paper was reviewed a score of the aggregated reviews could be assigned to it. Dall’Aglio (2006, p. 4) even proposes that this system could be used to rank reviewers in the process.

2.3.2.3 Peer review, publication and the web

As can be seen the academic community is well aware of certain deficiencies in the system and proposes models to change it.

¹See section 2.3.2.3 for more information about OAI
Attempting to include the web in the approaches to improve the system is not a new idea neither:

Wood (1998, p. 195) already stated that researchers would like to use electronic means in the peer review process to submit papers to publishers.

With the introduction of Web 2.0 however a lot of changes to the peer review and publishing process can occur: according to Kelty et al. (2008, p. 1002) the possibilities that arise with the new web change a lot of circumstances that were always taken for granted.

1. The process of publication is no longer hierarchical and capital intense. With the web as platform everyone is capable of publishing his ideas in blogs or wikis in a cheap and fast fashion.

2. Due to this self-publishing nature the volume of the material is growing even faster than before. However, by applying the same principles of the Web 2.0, this problem could be tackled by letting everyone review everything - however this approach would require a new form of reputation to ensure the validity of the review (Kelty et al. 2008, p. 1005).

Apart from these general changes the web has spawned certain new services for the publication of material and to make knowledge dissemination easier between researchers. These service include the Open Access Movement and the Open Archives Initiative (OAI).

The Open access movement is based around the idea that research results that are freely available will have a higher impact on further research² and that it is the right of every person to be able to access these results as they were funded by public subventions. In the spirit of these principles open access grants free access to scholarly articles (Jeffery 2006, p. 10).

There are two different types of open access: green and gold open access.

Green Open Access: This way the author of an article will self-archive the article in the form of the final draft that is delivered to peer-review. This way the

²According to Hull et al. (2008, p. 1) the assumption of greater impact is correct: open access articles are read and cited more often than non-open-access articles.
article can still be published in a journal but will be accessible for free in the archived form. In general green open access repositories are either thematic (based around a certain subject area - e.g. arXiv for physic publications) or institutional (a central repository for one institution).

**Gold Open Access:** In this case the author (or the author’s institution) pays a fee to the publisher and the publisher will make the article available for free.

The OAI was formed in 1999 in Santa Fee to provide new ways for knowledge dissemination in the web era. It is based around the idea to provide access to archives in order to lower the latency for research and to improve cross-discipline research (Lagoze and Van de Sompel 2001, p. 1).

To provide this interoperability they proposed the usage of metadata-harvesting and specified the OAI-Protocol for Metadata Harvesting (PMH).

### 2.4 User management

As can already be derived from the explanations about peer review and publishing there are many different persons involved in the process of publishing and peer reviewing a document.

These roles all possess different kinds of information and different powers (e.g. an editor can accept or reject a paper from publishing, whereas a referee is not allowed to know the name of an author in double blind refereeing).

These facts needs to be considered in the system and thus a form of access control and a system to provide a basis for trust needs to be established. This section will provide an overview over the common approaches in literature for both problems.

#### 2.4.1 Access control

To provide effective access control two steps need to be performed: authentication and authorization (Shah 2007, pp. 65-66).

³Archives in their interpretation means a repository of scholarly papers.
CHAPTER 2. LITERATURE REVIEW

Authentication: Authentication is the process of verifying the identity of a user to the system. Typically this includes a user name and a password. Safer variants can employ public or private key encryption (Halsall 2005, p. 652).

Authorization: Authorization determines which actions the user is allowed to perform in the system.

Authorization is most often based on different models that will be outlined in the following paragraphs. These models include Discretionary Access Control (DAC), Mandatory Access Control (MAC) and RBAC (Sandhu et al. 1996, p. 40).

MAC: In a MAC-based environment the access control is established via labels that are attached to users and objects and define the possible actions a user can exert on an object.

DAC: In a DAC-based environment the owner of an object determines who is allowed to access it.

RBAC: RBAC systems add a layer of indirection to the access control framework, by assigning permissions to roles instead of users (Nemeth et al. 2010, p. 108). These roles will in turn be assigned to users.

As RBAC is considered especially well suited for managing a large number of permissions it shall be highlighted in more detail (Nemeth et al. 2010, p. 108).

Sandhu et al. (1996, p. 40) point out that the RBAC is an access control model that supports the three security principles:

Least privilege: It is possible to assign only those privileges that are required to fulfill a certain task.

Separation of duties: It is possible to force mutually exclusive roles to be required to fulfill a certain task.

Data abstraction: The permissions that are established can be abstract concepts instead of the classic read, and write permissions.
A schematic RBAC model shall be provided to highlight the dependencies between the different concepts that make up RBAC:

![RBAC Diagram](image)

**Figure 2.2:** Basic RBAC model (modified from Sandhu et al. (1996, p. 41)).

As can be seen, a user can have multiple roles and each role can have multiple permissions.

However, depending on the current session it is possible for the user to only activate a subset of the roles that he actually possesses.

Apart from this basic model there are variations that can include role inheritance or constraints on sessions (thus not allowing to activate two roles simultaneously).

### 2.4.2 Trust

Another concept of great importance, whenever the interaction of different persons is concerned is trust.

Even though trust has been researched in many different fields to a great extend, this section shall only provide a short overview over the topic in respect to trust in virtual communities.

According to Dasgupta (2006, p. 449), trust can be defined as a belief that the “other party will refrain from opportunistic behaviour and will not take advantage of the situation”. This is considered of great importance in virtual communities, as
in such an environment social rules are considered to be less strict or even absent and thus no other system except trust exists that one can rely on.

In the context of virtual communities Lynch (2001, p. 16) offers a good definition of trust by defining it via an association between identity and behaviour: in this context an identity is required as a basis for making a person accountable and based on this accountability it is possible to observe a behaviour that will lead to trust in the identity.

This kind of trust can be considered as a tool for complexity reduction (Abdul-Rahman and Hailes 2000, p. 1).

Another facet of trust is the differentiation between interpersonal and system trust (Dasgupta 2006, p. 6):

Interpersonal trust is the trust based in another person and was discussed before.

System trust in comparison, is the trust in a “depersonalised system”: according to Dasgupta (2006, p. 6) this trust can be established by utilizing “protocols, certification, cryptography, authentication procedures and standards”.

### 2.5 Current systems

Compared to theoretical models and general approaches, this section will highlight the currently available systems that try to support the process of e-publishing.

It will especially highlight the features that are correlated with Web 2.0, publication and user management as well as peer review:

1. Does the tool support any Web 2.0 features?
2. Does the tool support publication management features (see subsection 2.3.1)?
3. Does the tool allow user management?
4. Does the tool support the process of peer review?

The comparison will be based on the work of Cyzyk and Choudhury (2008), Hull et al. (2008) and Goh et al. (2006).

The ePublishing tools that will be compared are: GNU ePrints, Open Journal System, DPubS and Fedora.
<table>
<thead>
<tr>
<th>Tool</th>
<th>Web 2.0</th>
<th>Publication management</th>
<th>User management</th>
<th>Peer review</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fedora</td>
<td>Paper submission. No collaboration.</td>
<td>Allows versioning (tracks changes to objects). CNRI handles.</td>
<td>Only two roles: administrator or anonymous user.</td>
<td>Not supported.</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison of e-Publishing tools.
It is obvious that most of the systems lack much of the Web 2.0 specific functions: even though the introduction of such features has already been discussed in the literature (see section 2.3) the introduction of alternative rating mechanisms or even tagging is still missing from all of these tools⁴.

Omittance of these features results in the foregoing of certain improvements:

**Knowledge dissemination:** With the addition of tags it would be possible to support easier resource finding, recommendation mechanisms or perform data mining on the tags to extract new information from the user-provided data (Hothe 2010).

**Quality assessment:** With a simple rating mechanism for the users it would be possible to harness the “wisdom of crowds” (see section 2.1). Or as Linus Thorvalds once said it: “Given enough eyeballs all bugs are shallow” (Raymond 1999, p. 8).

---

⁴A system that tries to incorporate these features seems to be in development, however the documentation was still lacking and no official release yet available: Sommarive (2010). Due to these reasons it was omitted in the overview.
REQUIREMENTS ANALYSIS

This section will describe all requirements the system has to fulfill.

The requirements definition according to Sommerville (2007, pp. 120-121) will be used as a basis for this chapter.

First the functional requirements will be discussed, followed by non-functional requirements and legal and ethical issues that need to be taken into account.

3.1 Functional requirements

This section will list the requirements that “describe what the system should do” (Sommerville 2007, p. 120).

Each requirement will be assigned a unique key for later reference.

3.1.1 Publication management requirements

These requirements are specific to the publication data the system should be able to handle.

DB01: The system shall provide an interface to store publication data.

DB02: The system shall provide an interface to retrieve the publication data.

DB03: The system shall store all provided publication data in a database.

DB04: The system shall be able to handle input data of publications in the BibTeX¹ format.

DB05: The system shall support the different types of publications available in the BibTeX specification.

DB06: Publication metadata might be subject to change, depending on retrievable metadata from external data providers. Refer to subsection 3.1.5 for further information.

¹http://www.bibtex.org/
By providing the users of the system with different input formats the system will be able to allow many users to participate in the system.

Moreover the system could be used as an open access repository, if not only metadata but whole documents are uploaded.

And, in association with the possibility to initiate a peer review this might lead to new quality controls by utilizing cross-platform referees: a facet that might lead to new possibilities to improve the quality of an article or even to new avenues of cross-discipline research.

### 3.1.2 User management requirements

These requirements will specify the different user management abilities.

**UM01:** The system will allow the creation, modification and deletion of new users for the provided service.

**UM02:** Users can have different privileges: the privileges will be derived from the publication procedure to allow a reconstruction of the peer review process (see subsection 2.3.1 and section 2.4 for detailed information about the peer-review and publishing process).

**UM03:** Privileges can be added, removed and edited.

**UM04:** Only administrators can add, remove and edit privileges.

**UM05:** A user can edit or delete his own user account.

**UM06:** A user can edit or delete the publications he added.

**UM07:** An administrator can edit or delete every publication.

**UM08:** The system shall store all provided user data in a database.

**UM09:** A user will have an associated reputation rating.

The implementation of these requirements will allow the user to specify the permissions needed in the system according to the current needs.
By implementing a RBAC system it will be possible to implement the different roles required in the peer review process.

Moreover it will make the adding of new roles easy for the administrator of the system.

The inclusion of a reputation mechanism will serve as a basis to built trust on: for helpful comments and active participation users will gain reputation that can be utilized as a quantified measure for their trustworthiness.

Moreover, it will mirror the reputation gain that researchers obtain by participating in the peer review process (see subsection 2.3.1) and it will have the function of an incentive to participate in the usage of the application (as reputation will help people getting notice, which according to Anderson (2006a, p. 74) is very important incentive for using Web 2.0 applications).

3.1.3 Web 2.0 technologies

This section will list the Web 2.0 technologies the system will employ.

WE01: The system will allow a user to add a tag to a publication.

WE02: The system will allow a user to add a rating to a publication.

The tagging and rating mechanisms will provide the core feature set of the application in respect to Web 2.0 technologies.

The rating mechanism can be used to implement a very abstract form of rating for publications that have not yet been peer-reviewed: this will allow even non-academics to rate publications if they desire and, if enough ratings are present, they can be used as a filter (see section 2.1 for more information about the harnessing of collective intelligence): this use for Web 2.0 methods as a post-publication filter was already hinted at in Anderson (2006b).

3.1.4 User interface requirements

These requirements are specific to the provided interface for the end user.

UI01: The system shall provide an interface to store new publications via manual user input. This interface will allow the input of all fields per hand: the
system will use the same fields as input that a \texttt{BibTeX} entry for the same type of publication would allow.

**UI02:** The system shall provide an interface to store new publications that will take a \texttt{BibTeX} string as input.

**UI03:** The system will provide an interface to store new users.

**UI04:** The system will provide an interface to edit user data.

**UI05:** The system will provide an interface to manage different privileges for a user.

**UI06:** The system will provide an interface to edit publication data.

**UI07:** The system will inform the user when incorrect input data was provided.

**UI08:** The system will display the data about a publication dynamically and add metadata to the publication (see subsection 3.1.5 for more information about metadata).

### 3.1.5 External interface requirements

These requirements specify what external services will be integrated into the system and what data will be provided to other services via the system.

**EI01:** The system will integrate an interface to the external CiteSeerX² service to acquire data from it.

**EI02:** The data acquired from the external service can be: publication data or metadata about a publication.

**EI03:** The system will use the OAI-PMH interface to retrieve the metadata.

**EI04:** Should the OAI-PMH interface be abandoned by CiteSeerX, the functionality will be lost, but the system shall remain functional apart from this.

²\url{http://citeseerx.ist.psu.edu/}
EI05: Publication information of the system shall be made available via Extensible Markup Language (XML)-format.

Utilizing a well-defined protocol to exchange data will allow easier integration of other OAI archives.

It might be considered to include an OAI-PMH interface to the data stored in the new system as well, to facilitate knowledge dissemination with other systems.

3.2 Non-functional requirements

This section will describe the non-functional requirements the system shall provide.

The system has to fulfill the following usability criteria:

UA01: The system shall provide an easy to use administration-interface. The usability aspect will be verified via user evaluation through a questionnaire and observation of task based activities.

Moreover it must fulfill certain safety, security and privacy requirements.

SE01: The access to different actions shall be restricted according to a user’s role (see subsection 3.1.2).

SA01: The system shall assist the user in entering data to prevent the entry of malformed data.

SA02: The system shall verify entered data.

SA04: The system shall obey the copyright law of the United Kingdom (see subsection 3.3.3 for detailed informations).

PR01: The private data entered by a user shall not be accessible by other users, if the user does not choose to reveal this information.

PR02: The system shall respect the rules laid out in the Data Protection Act from the British Government (see subsection 3.3.4 for detailed information).
Furthermore, according to the issued requirements of the user the implementa-
tion will be performed in an open-source programming language with MySQL
as the database backend. The system will use MySQL (version 5.1).

3.3 Professional, legal and ethical issues

This chapter will highlight different issues that need to be taken into account when
developing the application.

3.3.1 Professional and ethical issues

To guarantee high quality work and provide readers with a basis for trust, the
development of the application will follow the moral and professional code of
conduct outlined by Council (1992), where applicable.

Concrete points that will be taken into account are:

Moral code: The moral issues applicable to the development of the application
are:

1. The developed application will honor property rights and copyright
   claims (see subsection 3.3.3).
2. Proper credit for intellectual property will be provided in the application
   as well as the documentation.
3. The application will respect the privacy of its users, by adhering to the
data protection act of the United Kingdom (see subsection 3.3.4).
4. The application will honor confidentiality of data, by adherence to the
data protection act of the United Kingdom (see subsection 3.3.4).

Professional code: The professional issues applicable to the development of the
application are:

1. While developing of the application the developer will strive to achieve
   the highest quality possible according to his knowledge and skills.
2. The developer will respect existing laws that affect the developed ap-
   plication (see subsection 3.3.2).
3. The developer will analyse and evaluate the risks of the application in development.

### 3.3.2 Legal issues

When implementing the e-Publishing platform on the web, there are two main areas that need to be considered: the copyright law (as the system handles data associated with publications and may in the future allow the upload of whole documents) and the data protection law (as users will register to use the site and provide certain kinds of information during this process).

In the United Kingdom these two topics are covered by the law: the copyright law is codified in the *Copyright, Designs and Patents Act 1998* and the data protection law in the *Data Protection Act 1998*.

The main points of importance found in these two laws, will be highlighted in the following sections.

#### 3.3.3 Copyright law

The copyright law was created to protect the original works of an author from "unfair use".

The copyright owner must give his consent for the following actions to be performed with his work (Service 2000):

- "Copy the work.
- Rent, lend or issue copies of the work to the public.
- Perform, broadcast or show the work in public.
- Adapt the work."

Moreover the copyright owner has the right to be identified as the owner of a work.

#### 3.3.4 Data protection law

The data protection act issues the following rules that the application must adhere to (Crown 1998):
“Personal data shall be processed fairly and lawfully.

Personal data shall be obtained only for one or more specified and lawful purposes, and shall not be further processed in any manner incompatible with that purpose or those purposes.

Personal data shall be adequate, relevant and not excessive in relation to the purpose or purposes for which they are processed.

Personal data shall be accurate and, where necessary, kept up to date.

Personal data processed for any purpose or purposes shall not be kept for longer than is necessary for that purpose or those purposes.

Personal data shall be processed in accordance with the rights of data subjects under this Act.

Appropriate technical and organizational measures shall be taken against unauthorized or unlawful processing of personal data and against accidental loss or destruction of, or damage to, personal data.

Personal data shall not be transferred to a country or region outside the European Economic Area unless that country or region ensures an adequate level of protection for the rights and freedoms of data subjects in relation to the processing of personal data.”

With respect to these rules the application must provide the following features:

• An explanation for the collected data.

• Collect the minimum amount of data required.

• Delete the data when no longer needed, e.g. when a user account is deleted.

• The collected data will be protected with state-of-the art technical mechanisms (e.g. encryption of sensitive data with Advanced Encryption Standard (AES)).

• The servers hosting the application must be stationed within the European Economic Area or in a country providing adequate data protection.
This chapter will describe the overall design of the application (this is also referred to as *software architecture* in McConnell (2004, p. 43)).

The description will follow a top-down approach as portrayed in McConnell (2004, p. 82) and Sommerville (2007, p. 243):

1. Choose a model for the system’s organisation.
2. Decompose the system into subsystems.
3. Define a control style for the system.

Before the concrete description of the design, a short overview over the overall design goals shall be established. Moreover, the concrete technologies that will be used to implement the system will be described: this way it will be possible to highlight specific design decisions that relate to these concrete technologies.

### 4.1 Design goals

In the process of building this application the following design goals will be considered as essential goals that need to be achieved with the design (the selection is based on the list provided in McConnell (2004, p. 80) and Sommerville (2007, p. 243)):

**Complexity:** Even though extensible, the system’s design should be easy to understand.

**Loose coupling:** Connections between different parts of the system should be minimal. This aspect will be highlighted again in section 4.3.

**Extensibility:** The system should be easily extensible, without great changes to the system’s design.

**Security:** Especially with regard to requirement *SA04* and *PR01/02* security is an essential part of the system and as thus needs to be considered in the architecture already.
Even though there are more goals that could considered in the architecture, the aforementioned ones are of particular importance and will be considered in detail, when discussing the architecture and design of the system.

### 4.2 Utilized technologies

The development of the system will employ different technologies that might impact the design of the application. This section will introduce the technologies that will be used to implement the system.

#### 4.2.1 Programming language

When choosing a programming language for web programming, there exists a wide array of choices: common ones include Java, PHP: Hypertext Preprocessor (PHP), Ruby, Python and Active Server Pages (ASP).

From these options, ASP does not fulfill the requirement to be open-source, and - with the acquisition of Java by Oracle - the open-source nature of the Java Virtual Machine (JVM) is partially being undermined due to Oracle’s lawsuits against Google (Paul 2010).

The choice between PHP and Python or Ruby is mainly one of personal preference, even though many people consider PHP less clean considering the design of the language when compared to the other two languages (Cheever 2009; D’Angelo 2010).

Compared to Ruby, PHP’s and Python’s open-source-licenses are both Open Source Initiative (OSI)-approved ((OSI) 2011).

While PHP’s market share is still bigger than Python’s and Ruby’s, both languages have gained popularity in recent years (BV 2011) and many new sites are often based on Python (for example Quora¹ and Reddit²).

Apart from this fact Python offers some special features that might prove useful in the implementation of the system:

**First-class functions:** Python treats functions as objects. This way it is possible to pass functions as parameters or assign functions to variables.

---

¹ [http://www.quora.com](http://www.quora.com)
² [http://www.reddit.com](http://www.reddit.com)
Default parameters: Python functions allow the definition of functions with default parameters. This way the reusing of functions is easier.

Decorators: Decorators allow the transformation of functions. This can be used to execute certain code before each function by simply adding a decorator (e.g. verify that a user is logged into the system).

Multi-paradigm programming language: Instead of limiting itself to one programming paradigm (like Object Oriented Programming (OOP)), Python is a multi-paradigm programming language that allows the use of the OOP-paradigm, as well as the imperative and - to a certain extend - the functional programming paradigms.

Due to these facts, it was decided to implement the core platform in Python.

4.2.2 Frameworks and assisting technologies

When writing a web-application, Python offers a plethora of possible frameworks that can assist a programmer during the development process (P. S. Foundation 2011): following is an excerpt of the more full-featured frameworks (also referred to as full-stack frameworks):

Django: Provides templating, an admin-interface, an Object Relational Mapper (ORM)-layer and integrates a RBAC-system. Apart from this there exist a lot of plugins, and an extensive documentation on the website³.

Pyramid: A fairly new framework that supersedes the Pylons framework. Provides templating, but no ORM-layer, nor a RBAC-system.

web2py: Provides templating, an ORM-layer, a basic admin-interface and a RBAC-system. Provides moderate amounts of documentation on its website⁴.

When comparing these frameworks, Pyramid is the youngest of all (the initial release was on January 31, 2011 (Consulting 2011b)), whereas Django is the oldest

³http://www.djangoproject.com/
⁴http://web2py.com/
of the three mentioned frameworks (the first release of version 1.0 was in July 2005 (D. S. Foundation 2011c)).

Judging from the documentation available on the websites of each project, Django seems to provide the most documentation, whereas web2py and Pyramid seem to offer roughly the same amount (D. S. Foundation 2011d; Pierro 2010; Consulting 2011a).

Due to its features, the greater documentation, the existence of a variety of plugins and the overall maturity of the framework the project will be implemented on top of the Django framework, which leads to the following overall structure of the implementation technologies:

![Utilized technologies](image)

*Figure 4.1: Utilized technologies*

### 4.3 Subsystems of the application

This section will describe the choice of a model for the system's organisation, as well as the decomposition of the system into subsystems.

#### 4.3.1 System organisation

This section will first introduce the different possible ways to organise a system and then describe which approach seems most suitable for the application that is being developed.
According to Sommerville (2007, pp. 247-252) there are three major models that can be used to organise a system:

**Repository model:** The repository model allows subsystems to exchange information by sharing their data in a central repository. This model allows the efficient sharing of data, but it is necessary to implement multiple policies for security, recovery and backup.

**Client-server model:** The client-server model is based on an approach that organises the system as a set of services that are offered by a server and can be consumed by a client. Clients only need a network that allows them access to the services offered by the servers. This model has the advantage that the system automatically has a distributed architecture. However, to make use of the distributed nature the accessed data may need to be replicated between the servers.

**Layered model:** The layered model divides the system into a set of stacked layers. Each layer offers a set of services via an interface to the service above itself. One advantage of this approach is the possibility to change the implementation of a layer without changing layers that depend on it, as long as the interface remains unchanged. A drawback is the fact, that each layer should only communicate with each layer directly below itself and cross-layer calls should be prevented. This is not always possible and will introduce dependencies between subsystems that - according to the model - should not exist.

Taking into account that the architectural style of the web is a client-server model, the application will inevitably be based on this style: the application will offer a set of services that can be consumed by clients that send Hypertext Transfer Protocol (HTTP)-requests to a given Uniform Resource Identifier (URI).

Apart from this however, the system should be extensible and adhere to the principles of the Web 2.0: it should be easy for others to consume the service.

This means that the application needs to expose an interface that can easily be consumed: judging from the possible models the created application should
expose this via one interface that can be considered to be *layered* on top of the whole application (this interface will be a Representational State Transfer (REST) web-service - for more information see section 4.5).

### 4.3.2 Decomposition

When decomposing the system into its subsystems the decomposition can be performed along multiple levels: the first decomposition will be into subsystems or packages (McConnell 2004, p. 82).

The subsystems required by the system to operate correctly are the following:

**User interface:** This subsystem will provide the access point to the application and provide the requested data to the user. It will either return Hypertext Markup Language (HTML) or XML data - depending if the application is called as a web-service or via its admin-interface.

**Database:** This subsystem will handle the database calls. Due to the usage of an ORM mapper this subsystem is hidden in the used web-framework.

**Business logic:** This sub-system will implement required business-rules and application logic, like insertion, search or calls to external sites.

The relationships between the different subsystems can be seen in the following diagram:

![Diagram of subsystems](image)

**Figure 4.2:** Decomposition into subsystems.

To further decompose these subsystems, either an object-oriented approach or a function-oriented approach can be used (Sommerville 2007, pp. 253-254):
Due to the fact that Python is used to implement the system and the language
allows the usage of both paradigms, the paradigm most suited to each problem
will be utilized.

The concrete decomposition into classes and routines will be described in section 4.5.

4.3.3 Control style

The control style needs to be defined to allow the system to work: it defines how
sub-systems are called and how control needs to flow between these systems, to
generate the responses required by the user (Sommerville 2007, p. 256).

There are two major control styles:

Centralised control: This style defines one sub-system as a controller that will
receive all calls to the system. It can either be employed via a call-return or
a manager model.
The call return model uses a top-down approach to call required sub-systems
to fulfill a request.
The manager model is used in concurrent systems to start sub-systems based
on system-state.

Event-driven control: This style is based on external events that will trigger the
execution of a sub-system.

Judging from the decomposed sub-systems, the style of control used for the
application should be a centralised control style based on the call return model:
the user-interface sub-system will accept all incoming requests from clients and
distribute them to the correct system. After the required processing is finished,
the results of the call will be returned to the client.

4.4 Database

This chapter will describe the design of the database system.
4.4.1 Abstract database design

To create the database design, the process being followed is based upon the suggestions found in Hernandez (2003) and Stephens and Plew (2000). It will consist of the following steps:

1. Define a mission statement
2. Analyse existing databases
3. Create data structures
4. Determine and define business rules
5. Review data integrity

4.4.1.1 Mission statement

The mission statement of the database design phase shall declare the purpose of the data in general terms (Hernandez 2003).

With regard to the aforementioned requirements specified in chapter 3 a suitable mission statement for the database is:

“To maintain the data generated by the system and it’s users that is required to offer the publication and quality control services expected.”

Apart from the general mission statement it is suggested to provide an overview over the different mission objectives that describe specific tasks that need to be supported by the maintained data.

A short overview over these objectives shall be provided in the next subsubsection. This will ensure that no important tasks are forgotten and can not be fulfilled by the collected data.

4.4.1.2 Mission objectives

Each mission objective will represent a single task and not provide too many details. The objectives are based on the requirements found in chapter 3 on page 23.

- The data for a publication needs to be stored.
- Reference material associated with a publication needs to be stored.
- Comments associated with publications need to be stored.
- Tags associated with publications need to be stored.
- Ratings associated with publications need to be stored.
- Users need to be stored.
- Users can be non-verified or verified⁵.
- Roles needed to support the peer review process need to be stored.
- Permissions associated with roles need to be stored.
- A user’s esteem needs to be stored.

4.4.1.3 Analyse databases

To perform an analysis of former databases a different approach than proposed by Hernandez (2003) shall be used: as no previous database is available for the current system the databases of open-source-systems with similar functionalities shall be analysed.

To analyse the storage of publications, the database-schemas of CiteSeerX⁶ and the Open Journal System (OJS)⁷ shall be used as a basis for the analysis.

To study approaches how to store ratings and comments, as well as the implementation of a voting system the data sources for Stackoverflow⁸ and Reddit will be referenced.

---

⁵A verified user did provide some sort of validation to prove his or her identity.

⁶The source code including the data schemas can be obtained from http://sourceforge.net/projects/citeseerx/.

⁷The source code including the data schema in an XML representation can be obtained from http://pkp.sfu.ca/ojs_download.

⁸As Stackoverflow is not an open source application the early data model referenced on http://sqlserverpedia.com/wiki/Understanding_the_StackOverflow_Database_Schema will be referenced.
CHAPTER 4. DESIGN & SOFTWARE ARCHITECTURE

CiteSeerX & OJS database: The CiteSeerX database can be used as a general example for a publication database including a table for papers, authors, keywords and citations\(^9\). Apart from these tables it also includes a table to store custom tags - an approach required in our application as well.

The OJS database is another example of a typical publication management database, including tables for: journals, articles, comments, notes and keywords.

Apart from these tables the OJS-database-schema also incorporates a table to store different roles in the database.

Stackoverflow & Reddit: The analysis of Stackoverflow, Reddit and other sites is used to obtain an understanding how these sites implemented voting and rating mechanisms into their database schemas.

Stackoverflow uses tables for Users, Posts, Comments, Badges\(^{10}\), Votes and Vote-Types. These tables fulfill the requirements for a normalized database.

Compared to this approach, Reddit uses only two tables: Thing and Data\(^{11}\). This approach is called Entity-Attribute-Value and is used to guarantee high flexibility in the database (See Nadkarni (1999)).

Conclusion: Judging from the different approaches, the usage of normalized tables to store the data for publications, authors and associated information seems to be prevalent.

Taking into account that this will also lead to less redundancy and help in the prevention of data-inconsistency and wastage of storage space (Elmasri and Navathe 2003, pp. 15-16), makes it a suitable approach for storing the general information associated with publications and authors.

The Entity-Attribute-Value tables might provide useful mechanisms to allow the storage of further information in the database that might not be anticipated beforehand - thus allowing users to store further information about publications without limiting them to a certain selection of fields. This approach will be used

\(^9\)A reverse engineered database schema for CiteSeerX can be found on page 85.

\(^{10}\)Badges are small awards users can earn when participating in Stackoverflow.

\(^{11}\)See http://www.thinkvitamin.com/code/steve-huffman-on-lessons-learned-at-reddit/.
in special Entity-Attribute-Value tables that are linked to the publication and user tables, thus providing the possibility to add more information to those entities.

4.4.1.4 Create data structures

The creation of the data structures will mainly be based on the reference data schemas of Analyse databases on page 39, the overall project requirements defined in the Research Report and the BibTeX reference (Patashnik 1988).

The database can be divided into two big blocks - one to manage the users of the system and one to manage the publications and their associated data.

**User management:** The user management will be based on a RBAC-model and encompass tables to store Roles, Permissions, Users and a mapping of currently active roles for a user in a Session.

Apart from these tables it is necessary to store a Rating (in the data model this is referred to as Esteem to distinguish it from the ratings for publications) for a user (requirement um09).

Moreover the fact that two types of users are supported needs to be included in the schema as well - in this system it will be included by storing the fact if a user is authenticated in an is-authenticated flag.

**Publication management:** When defining the tables needed to store the information for publications, requirement db04 is of great importance - to fulfill this requirement, the BibTeX format for publications needs to be analysed: a definition which entries are possible can be found in Patashnik (1988, pp. 7-11).

Apart from these attributes it is necessary to store accompanying material for publications as well (like references or data sets).

Apart from the tables to store the direct publication information it is necessary to store information about the Author, Tags (requirement we01) and Ratings (requirement we02) for a publication. Moreover Keywords need to be stored in the database as well.
**Keywords & Tags:** The difference between keywords and tags will be that keywords can only be defined by the person that added the publication to the system, whereas tags can be added by every user of the system.

For storing the author information the normalized tables approach from CiteSeerX will be used, to reduce redundancy and allow easier maintenance.

Moreover metadata retrievable from other sites should be referenced from the correct publication as well. As metadata retrieved from other sites is likely to change, the information will not be persisted in the database. The information should rather be added dynamically, when a query for a certain publication is issued.

**Peer reviewing:** The peer reviewing component will be stored in the PaperGroup table that provides a place to store editors, referees and publications.

While a publication is under review it will only be accessible to users that are part of one of these two groups.

### 4.4.1.5 Determine and define business rules

The commonly used mechanisms to address business rules in database design include *primary keys, foreign keys, unique* and *check constraints, data types* and data precision as well as *null* and *not null* definitions (Stephens and Plew 2000, p. 297).

Based on the rules specified in the BibTeX documentation, the following constraints can be created:

In Table 8.2 on page 82 the **bold** fields mark required attributes for a given publication type and need to be provided, whereas the others are optional attributes: whenever a user submits a certain publication type all required attributes need to be filled in.

Apart from these rules the stored data concerning the users of the system needs to adhere to the standards of the United Kingdom (UK)’s data protection law: see section 3.3 of the Research Report for more information. This includes
the encryption of personal user information, as well as security measures to prevent unauthorized access to the data (these topics will be covered in detail in section 4.5).

Moreover the security issues mentioned in Elmasri and Navathe (2003, pp. 731-740) need to be addressed as well:

**Loss of integrity:** The data in the database needs to be protected from unauthorized access and modification - in this case the proper rights to modify data will be issued based upon roles (Elmasri and Navathe (2003, p. 733) refers to this as a *mandatory security mechanism*).

**Loss of availability:** The database is no longer accessible for authorized systems and users.

**Loss of confidentiality:** Refers to the aforementioned fact of disclosing confidential information to unauthorized sources.

These three problems can be countered by employing *access control, inference control, flow control* and *encryption*.

### 4.4.1.6 Data model

Having created the data-structures the data-model can be displayed in an Entity-relationship model (ERM) data model that can be found on page 44.

### 4.4.1.7 Review data integrity

Data integrity in the database can be handled by enforcing *entity integrity, domain integrity* and *referential integrity*.

**Entity integrity:** It ensures that each table has a primary key row that holds unique values and thus can be used to uniquely identify a row of the table.

**Referential integrity:** It ensures that other entities that are referenced exist. This is also known as foreign key constraint.
Figure 4.3: Data-model for the publication management system.
4.4.2 Database implementation

This section will highlight the differences between the theoretical model of the database and the concrete model due to the technologies employed:

As the application will be built on top of the Django framework, the user authentication system will use the one that is distributed with Django¹².

Apart from this, the only difference is that the tables will be created using the ORM mapper distributed with Django instead of creating the database-tables by hand.

When creating the tables it is necessary to make sure that the tables will be created with the InnoDB database engine and not the standard MyISAM engine of MySQL: compared to MyISAM, the InnoDB engine does support foreign key constraints, whereas the MyISAM engine does not.

4.5 Application design

This chapter will focus on the application design. This includes all sub-systems derived in subsection 4.3.2.

The focus lies mainly on the theoretical design of these sub-systems - the concrete implementation details will be specified in the following chapter 5.

The first section will focus on the design of the external interface of the web-service.

4.5.1 Web service design

The application will provide an interface for other applications to consume, in the form of a RESTful-web-service.

The design of this web service will be described in this chapter, after an explanation what a RESTful-web-service is and why the REST approach will be used.

¹²The tables in the datamodel on page 44 already resemble the ones from the Django authentication framework.
4.5.1.1 REST-web-services: an overview

When building web services, there are two possible approaches that can be taken to implement one: the web service protocol stack and REST.

This section is dedicated to describing the two approaches and explaining why the REST approach was chosen for this application. Therefore, the ideologies that the two approaches are built on will be explained and advantages and disadvantages of each approach will be highlighted.

The web service protocol stack: The ws-protocol-stack is based on multiple specifications that should be used together to build a “big web service” (Richardson and Ruby 2007, p. 15). The main technologies are the Web Service Description Language (WSDL), the Simple Object Access Protocol (SOAP) and ws-Policy among others (like ws-Security).

The basic protocol these big web services are based on is SOAP: SOAP is a protocol that - like HTTP - specifies an envelope that can contain headers, as well as metadata and data (Webber et al. 2010, p. 367).

Another similarity between SOAP and HTTP is the support for intermediaries that could provide caching for requested messages: this useful feature is, however, eliminated by the fact that SOAP tunnels all messages through the HTTP-request that can not be cached (Webber et al. 2010, p. 377).

In contrast to HTTP, SOAP behaves more like a messaging middleware: it defines an envelope, the way to transfer the message (most often HTTP), but does not provide any application logic. How a message is interpreted is left entirely to the service.

Moreover, SOAP only provides a SOAP-fault to indicate an error condition.

To describe the different services that can be consumed via SOAP messages, WSDL is used: in most scenarios WSDL is used an Interface Description Language (IDL).

This leads to one problem of WS-based web services: utilizing WSDL to describe a web service will lead to a tight coupling between the web service’s implementation and the service contract.

Taking into account that many WSDL web services are implemented with tool
support in Java or C#, the coupling will even be enforced between the web service's implementation and the end user: a fact that leads to a violation of the encapsulation principle, as a user depends on an exact implementation of the web service that was generated through a certain tool-chain (Webber et al. 2010, p. 383).

**REST**: Compared to SOAP, REST is an application protocol that is based on the principle to use the web as a “connected set of resources” (Slobojan 2010, p. 77) or, as stated in Webber et al. (2010, p. 12), to use the web as a “distributed hypermedia application whose linked resources communicate by exchanging representations of resource state”.

The defining characteristics of a web service built as a RESTful web service are the following (Richardson and Ruby 2007, pp. 9-11; Slobojan 2010, pp. 1-8):

- A RESTful web service exposes data as resources.
- The information about an action to be performed on a resource is encoded in the HTTP method: the typical methods that are used are:
  - **GET**: Will query information about a resource without changing it. GET is an idempotent request (this means that subsequent GET request will leave the web service in the same state).
  - **PUT**: Will modify an existing resource or create a new resource by specifying its ID. PUT is an idempotent request: if a resource is changed twice with the same request it should be in the exact same state. This leads to the practice that PUT request should not make relative changes to a resource.
  - **POST**: Will create a new resource, whose Uniform Resource Locator (URL) is set by the web service. POST is not an idempotent request: if two identical POST request are send to a web service, two resource instances will be created.
  - **DELETE**: Will delete a resource (either by making it inaccessible through the web service or really deleting the resource completely). DELETE is an idempotent request.
OPTIONS: Will return a list of possible methods that can be invoked on the resource. OPTIONS is an idempotent request.

- A resource can have multiple representations.
- Everything has an ID.
- Resources are linked together (also referred to as “hypermedia as the engine of application state”): it is possible to transition through a REST web service by following links - this concept is similar to the functioning of a state machine.

When building a RESTful web service this web service will also harness the advantages of the web’s architecture (Webber et al. 2010, pp. 12-18; Richardson and Ruby 2007, pp. 84-103):

**Scalability:** HTTP allows caching for certain requests like GET, allowing horizontal scaling.

**Loose coupling:** The web does not provide integrity-checking mechanisms for resources, thus preventing tight coupling with clients. Instead status codes can be used to indicate state-changes to a resource.

**Consistency & Uniformity:** Uniformity refers to the fact that the web restricts possible operations to a small number (the HTTP verbs) with pre-defined semantics. If these are applied according to their specification, resources that can be accessed with them act consistent.

**Addressability:** This refers to the fact that all resources exposed in a RESTful manner are accessible via URIs that can be transferred and reused.

**Statelessness:** Each request of the web is stateless and is considered in isolation: for a RESTful web service this means that each request must contain all information to fulfill a request. Due to this attribute it is possible to employ “load balancing” on the web.

**Multiple resource representations:** By allowing HTTP-requests to contain ACCEPT and CONTENT-TYPE headers it is possible to have multiple representations per resource that can be served depending on a request.
Conclusion: When comparing the ws-protocol-stack to the REST approach with regards to the goals specified in section 4.1, it is obvious that the tight coupling of the ws-stack does not fulfill the loose coupling goal.

Moreover, with REST being based on the well-known HTTP protocol the usage of the service will be less complex for developers familiar with the web.

To provide necessary security features, both solutions could use the possibilities provided by HTTP.

4.5.1.2 Designing a RESTful web service

The following section will now describe the design of a RESTful web service that is based on the datamodel of section 4.4.

It will follow the process described in Richardson and Ruby (2007, p. 109) that consists of the following steps:

1. Split the data into resources.
2. Name the resources.
3. Expose a uniform interface.
4. Design accepted representations.
5. Design served representations.
6. Define the flow of control.
7. Define error conditions.

Split data into resources: Richardson and Ruby’s (2007, p. 112) state that “anything interesting enough to be the target of a hypertext link” should become a resource. There are three possible types of resources (Richardson and Ruby 2007, p. 113):

Predefined one-off resources: These resources are top-level directories of other resources - in case of the e-publishing system this would be a resource that points to other stored resources: for example a directory that points to lists of publications, authors and other stored resources.
CHAPTER 4. DESIGN & SOFTWARE ARCHITECTURE

Resources for every object exposed: These type of resource is exposed for every table in the database that can be seen by the user.

Resources representing algorithm results: These type of resources are exposed for each kind of resource that is searchable by the user: publications, authors, tags, comments, peer reviews, users.

For each of the defined resources the service can provide a list or the concrete data associated with one instance of the resource.

Certain resources will allow searching - the search results will be lists containing links to the individual resources.

Name resources: The resource names will describe the URIs of the resources that can be consumed via the service.

Each URI will begin with the service’s URL that will be abbreviated with the placeholder {service}.

The web service’s root URL at http://{service}/ will contain a list with links to the other resources: e.g. http://{service}/publication/. It is possible to attach certain exposed resources directly to the base-URL which will return a list of all resources of this type: these resources include: publication, author, keyword, papergroup, reference_material, researcharea, tag and user.

For example, querying the URL http://{service}/publication/ will return a list of all publications.

When a URL is followed by an id (e.g. http://{service}/publication/{publication-id}) the publication associated with that id will be affected.

To allow the user to search the stored data the web service will allow the searching of certain resources¹³: like Richardson and Ruby’s (2007, p. 122) example the web service will expose the search interface via query variables that must adhere to the following syntax: {attribute}={value}&{attribute2}={value}.

To search a publication with a title that contains the word code the URL would be: http://{service}/publication/title=code/.

¹³Currently only authors and publications can be searched
Expose a uniform interface: This section will define the methods that a client is allowed to invoke on the different resources:

All resources that will return lists of links can be queried via GET and POST requests: when querying a resource via a POST request, the URL will function as a factory URL that will create the new resource and assign its address that will then be returned to the client in the HTTP-location-header.

All resources that work on a single object can be subject to GET, PUT and DELETE requests.

As a consequence of the fact that the web service will create the unique IDs of each resource, a client needs to create a new instance via POST and can then modify it via PUT requests. The creation of resources via PUT by setting the ID as a client is not supported.

Design accepted representation: The accepted representations to insert or modify data to the service will be the same as the one’s returned by the web service (see Design served representation: for more information) with one exception (Richardson and Ruby 2007, p. 151): when a new publication should be added the client can either send the same representation that is returned from the web service or it is possible to create a publication by sending a representation of it in BibTeX format.

Design served representation: The initial web-service will serve (and accept) XML data. The representations served for each resource can be found in section 8.3.

Flow of control: To allow an easy programmatic use of the web service, the flow of control for the user side will be handled mainly via HTTP-response-codes that will reflect the answer to a request. The following status codes will be used:

200 - ok: The request was understood, carried out correctly and the response is returned in the message body. This status code will be returned when the client requests resources via GET, HEAD or OPTIONS.
201 - CREATED: A resource was successfully created - the location of the resource will be returned in the LOCATION header.

204 - NO CONTENT: This status code indicates that the request was processed correctly but no answer is generated. This will be used when deleting a resource.

400 - BAD REQUEST: A general error code that indicates that the client send an incomprehensible request to the server. Will be sent when the other status codes in the 400 range do not apply.

401 - UNAUTHORIZED: The client was not authorized to perform the desired action.

403 - FORBIDDEN: This code will be returned when a client does not possess the necessary permissions to access a resource.

405 - METHOD NOT ALLOWED: This code will be returned when the client issues a HTTP-method that is not supported by the resource that is the target of the request.

415 - UNSUPPORTED MEDIA TYPE: This status code will be returned when the client does not send a valid representation of data to the service, or when the client does not accept one of the possible representations for the data the web-service could return (e.g. the client does not set the ACCEPT header in a request).

500 - INTERNAL SERVER ERROR: The general status code to indicate a server error.

Error conditions: The error conditions the service has to handle correspond to the error codes of the 400 range: requesting invalid data, sending invalid data or requesting data when missing the necessary authentication.

4.5.2 Module design

After designing the interface to the web service, this section will explain the overall design of the view and the business logic modules.
Therefore the modules are decomposed further.

4.5.2.1 View module

Admin interface: The admin interface provides web-pages that allow the direct manipulation of data in the system. This means the admin interface is directly accessing the ORM mapper to access objects without any intermediary business rules interfering.

The admin interface provides HTML pages that allow an administrative user to add, edit or delete entries for all resources present in the database.

Due to the fact that this module is based on the administration module provided by Django that will be modified where necessary no upfront design is necessary.

Web service views: The web service views implement the web service functionality described in subsection 4.5.1.

When designing the views that correspond to the resources identified in section 4.5, it is necessary that each resource is capable of defining the allowed HTTP-methods that might be called on it, as well as the ability for each view to return different kinds of representations for each resource.

To allow this and provide the ability to dispatch the correct HTTP-method to the appropriate method, the following design was chosen:
The `RestView` superclass will identify the HTTP-method used to call a given resource and dispatch the call to the appropriate method of a given subclass.

In this regard the `RestView` class serves as a manager (refer to subsection 4.3.3 for more information) for its subclasses.

The subclasses that can be addressed implement only the possible HTTP-methods and make them known via the allowed_methods list. The concrete implementation of each method resides in a function with the appropriate name (e.g. GET).
4.5.2.2 Business logic module

The business logic module contains four sub-modules that can be used by the views:

**Insertion module**: The insertion package will allow other parts of the program to insert new objects into the database. Its design is depicted in the following diagram:

![Diagram of Insertion module](image)

Figure 4.5: Insertion module.

To allow the insertion module to be extensible, it provides an abstract inter-
face to other classes that identifies which methods will be exposed by this system. This interface acts as a facade for other systems.

This interface must then be implemented by concrete inserters (e.g. XMLInserter) that can handle a given content-type.

To acquire a concrete inserter a factory-function is provided that will return an inserter based on the type of content that should be inserted. This way more inserters can easily be added later on, without requiring changes in the calling code.

Search module: The search module allows the querying of the database with certain constraints that need to be fulfilled: it allows the reduction of results to a subset conforming to certain matching rules.

It provides method to search publications, authors, keywords and tags.

Moreover it is used to obtain recommendations based on existing resources: it can, for example, return tags or keywords that were used in conjunction with another keyword, or users that might be related to a publication.

This functionality can be used to limit the search for peer reviewers to those relevant for a given publication.

External query module: This module encapsulates logic that allows the system to consume other services and integrate the data at runtime.

The external query module can be used to built objects that can query OAI-PMH enabled services.

The obtained data will then be dynamically added to objects.

In the beginning this system will provide a bridge to the CiteSeerX metadata service to showcase the functionality.

Access module: The access sub-module exposes special checks for other modules to verify that a certain authentication-condition is valid.

It can be used to perform fine-grained access checks for certain functions.

Moreover an internal module to parse BIBTEX files is part of the business layer as well.
This chapter will describe certain details of the implementation that were discussed in the previous chapter¹. Moreover it will highlight details how the implementation ensures the privacy, safety and security constraints that were issued in section 3.2.

5.1 View

This section will describe how the view handles the creation of the responses to requests and how it is possible to extend this mechanism with new representations.

5.1.1 MVC and Django

Django is a MVC-framework that distinguishes between models, views and templates as compared to the standard distinction of model, view and controller found in Gamma et al. (1995, pp. 14-16).

The view is a function that will collect the information to display and pass it to a template that will render the information, using Django’s templating language to fill in dynamic information on the fly.

To allow this replacing mechanism to work, Django templates provide special tags that will tell the templating engine to replace a variable with the value specified in the tag: {{ value }}.

Moreover the templates allow statements for flow control, like if-else- and for-statements. These need to be enclosed in {% statements %}-tags.

To know which view to call, Django provides a file that contains the necessary URL-routing directives called urls.py.

A schematic depiction of the complete request-dispatching process looks as follows:

¹The full source code of the application is provided on the accompanying DVD in the Source Code folder.
The system uses templates for each resource that is provided by the web service (all templates can be found in subsection 8.3.1): therefore the appropriate view provides the template’s name and a dictionary of values that can be referenced in the view to fill in the appropriate fields.

```python
1 template = get_template(template_name + dotted_suffix)
2 response = template.render(Context(dictionary))
```

Listing 5.1: Calling a template in the view.

This way adding new representations is decoupled from the code: to create a new type of template simply adding a new subdirectory to the templates folder of the web service is enough. Inside this folder all templates for this type can be placed. The only restriction is that these templates need a suffix corresponding to the response type: if a new resource would be created for Javascript Object Notation (JSON), all templates would need to end in .json so the service could identify them as those matching the JSON requests.

**Template or Parser:** When creating the responses for the client, there were two major approaches to be considered:
• Writing a parser for each format that can be returned.

• Utilizing templates and create a new set of templates for each format.

The advantages of a parser would be that adding a new resource would be trivial with a new call to the parser. However writing a parser that generates exactly the needed output is a time-consuming activity.

Compared to this adding a new file for one new resource does not take too much time and is trivial. Moreover, utilizing the provided features of Django’s templating language it is even possible to add flow control to the templates. Apart from this utilizing the templates integrates better with the system and is less error prone.

To implement the administration interface it is enough to add the models that should be shown on the administration interface (and modifications to the appearance of the HTML documents) to a file called admin.py. Some pictures of the application are provided on page 87.

5.1.2 Privacy, safety and security

The view helps in fulfilling the non-functional requirements for security and privacy by providing the following mechanisms:

Security: The view restricts access to resources by utilizing a mechanism to require a login for certain actions: these actions include most PUT, POST and DELETE requests.

To limit the access to these methods the framework provides a Python-decorator called @login_required that will check if a user is logged in and redirect the client if this is not the case.

Privacy: When creating a user account via the web service it is not necessary to provide any more information than the username and a password. This way the system collects the least possible amount of information a user is willing to provide.
Apart from these implemented mechanisms, the Django framework already provides protection against Cross-site request forgery (CSRF)-attacks (D. S. Foundation 2011b):

**CSRF:** These kinds of attack attempt to hijack a user’s sessions and perform actions masked as the user.

Moreover all values in templates will be escaped by the framework before they are sent back to the client, thus preventing Cross-site scripting (XSS) attacks:

**XSS:** These attacks attempt to inject client-side scripting (e.g. JavaScript code) into a website, thus circumventing the same origin policy². When the code is run by the client’s browser it could for example read the client’s cookie or send confidential information to the server of a third party.

### 5.2 Business Logic

The business logic layer performs the processing steps that require more logic than simple interaction with database:

simple interaction (querying state) are performed by the view³ itself as is seen in the diagram in subsection 5.1.1.

#### 5.2.1 Insertion

The insertion module provides insertion classes that are used by the web service to insert user provided data.

To allow a user to insert data with different content-types, the module provides a factory method that takes a content-type as parameter and will return an appropriate inserter for the given content-type:

³View is used in the Django terminology here, whereas a view contains more logic than simply displaying data. A Django view is seen as a controller by some users of the framework.
def get_inserter(content_type):
    """Returns an inserter based on the provided content-type.

    Factory function that returns a fitting parser based on the content-type that
    should be inserted.

    Attributes:
    - content_type: the content-type the user wants to insert
      - should be either xml or json.

    Returns:
    - inserter: an inserter that can be used to insert the given format."""
    if 'xml' in content_type:
        return XmlInserter()
    elif 'json' in content_type:
        return JsonInserter()
    else:
        raise AttributeError("No inserter for specified value present: %s." % (content_type))

Listing 5.2: Factory method to obtain an inserter

This way it is possible for each inserter to perform its own validations of the provided format (e.g. validating the well-formedness of an XML document).

The administration interface on the other hand does not use these routines, but is only limited by the validators used in the model itself: this distinction is, on the one hand, enforced by the framework but on the other hand this allows special handling for administrative users. Should the administrator’s require special sanity checks it is easily possible to add them via the Django framework: see admin validation⁴.

⁴https://docs.djangoproject.com/en/dev/ref/contrib/admin/#adding-custom-validation-to-the-admin
Bibtex insertion: To insert publication from BibTeX, the insertion module uses a BibTeX parser that was written from scratch utilizing the pyparsing module. The parser is contained in the BibtexParser class and implements a parser based on the BNF in listing Listing 8.27 on page 106.

5.2.2 Search

The search module is implemented in a modular way to allow adding search functionality for new objects easily: the core algorithm builds a search query that will attempt to match a key to an attribute on an object and the value will be matched according to an SQL like statement. If the attribute does not exist the search will only fail when performed, as the building algorithm does not know for what object the search is constructed:

```python
def build_query(search_query):
    """Build the query for a given dictionary.
    Will create a Q object for each query and return a list of these queries.
    Attributes:
    search_query: a dictionary with search_term: search_value pairs.
    Returns:
    query: a list of Q-objects containing the query.
    Raises:
    AttributeError: when a search_value is not present for a search_term."""
    query = []
    for key, value in search_query.items():
        if value:
            # If an id is provided an exact match is required.
            if key == 'id':
```
CHAPTER 5. IMPLEMENTATION

63

```python
q = Q(**{'%s__iexact' % (key): value})
else:
    q = Q(**{'%s__icontains' % (key): value})
query.append(q)
else:
    raise AttributeError("Invalid value for keyword %s: %s" % (key, value))
return query
```

Listing 5.3: Building a generic search in Django

This way searches based on the attributes of a simple objects can be constructed with dictionaries of the form `{'attribute': 'value'}`.

### 5.2.3 External queries

The external query module provides the class `OaiPmhDecorator` that can be used to construct an object that can query web services that adhere to the OAI-PMH protocol.

It will built the necessary URLs via a function that is passed as a parameter to the constructor: this way even services that have complex URL patterns can be queried by the service⁵.

As soon as an answer is retrieved, the attributes returned from the service will be added to the object that the query was made for. This feature uses the fact that Python allows the runtime modification of objects via it's `setattr` function (see Documentation⁶).

Even though this would allow the extracting of further information for publications, there is one severe drawback to the external query module: when it is used to query the CiteSeer service responses can easily take longer than 10 seconds to create, rendering the web-service too slow for productive use.

---

⁵This implements the strategy pattern (Gamma et al. 1995, p. 349) in a language that supports first-class functions.

⁶[http://docs.python.org/tutorial/classes.html](http://docs.python.org/tutorial/classes.html)
5.2.4 Access control

The access module provides certain functions that can be used to check certain conditions.

These functions include e.g. a verification if a user is a referee, or editor in a certain group. These functions should be used to allow more fine-grained access-control than a ROLE-based model.

5.2.5 Signals

Apart from the obvious modules that provide functionality, some actions are only performed when certain objects should be saved:

- On saving a new user, an associated esteem entry needs to be created.
- When a vote is cast on a comment, the commenting user's esteem will change.

Instead of performing these action in the insertion phase (which would link them to concrete inserters), they have been decoupled from the rest of the system by using a signal mechanism: whenever an object of type user or vote is saved the appropriate actions described above will be performed.

This functionality uses the Django signals\(^7\) module.

5.2.6 Privacy, safety and security

The business logic module helps in fulfilling the goals for safety and security requirements.

Both goals are supported by extra validations that can be performed while inserting or updating new objects into the system.

An example of these special validations is the fact that publications that are entered will be checked according to the BibTeX rules: if a publication is inserted via the BibTeX-insertion, but is missing a field according to the specification, it will be rejected by the service.

\(^7\)https://docs.djangoproject.com/en/dev/topics/signals/
CHAPTER 5. IMPLEMENTATION

5.3 Database layer

All database interaction in the application is performed through Django’s ORM mapper: this requires that all model classes are defined in the file models.py.

To allow the database to be extensible for later developers the final database was set as an initial migration for the South⁸ database migration tool: this way changes to the model-file can automatically be migrated to the database without the need to write complex SQL statements to alter the database, or even drop all tables and recreate them.

5.3.1 Privacy, safety and security

The database implements features that fulfill all three non-functional requirements:

Safety: By validating inserted data against the integrity rules (see Review data integrity), the database prevents insertion of malformed data.

Moreover, the ORM mapper includes escaping mechanisms to prevent the use of SQL-injections.

SQL-injection: This attack is focused around the manipulation of request parameters (most often for POST and GET requests) to execute client-generated database statements (Holovaty and Kaplan-Moss 2009, p. 342):

```python
1 def user_contacts(request):
2     user = request.GET['username']
3     sql = "SELECT * FROM user_contacts WHERE username = '%s';" % username

Listing 5.4: SQL-injection vulnerability
```

If an attacker now provides the string `'; DELETE FROM user_contacts WHERE `a'` = `a'` as username-parameter all users in the system would be deleted.

⁸http://south.aeracode.org/
Security: All passwords are stored as salted hashes.

**Salted Hash:** A hash is the output of a function that computes a fixed-length bit string from an arbitrarily long piece of text. This technique alone already helps in securing secrecy (Tanenbaum 2002).

Due to the fact that the hash-function must result in the same hash for the same input, there exist rainbow-tables of pre calculated hashes for common words that can be used to crack the input.

To eliminate this possibility a salt is added to the hash-function’s input: it is a random number of bits that will modify the output and is normally stored alongside the hash-function. A salt renders rainbow-tables practically useless.

Privacy: As for security, the passwords of user’s are encrypted via a salted Secure Hash Algorithm 1 (SHA1) hash.

**Current System:** The system is currently hosted and accessible on the infrastructure of http://gondor.io under the web-address http://du865.o1.gondor.io/.

The administration interface can be found under http://du865.o1.gondor.io/admin and the login is:

- Username: django
- Password: heriotwatt2011

Moreover, the accompanying DVD provides an Ubuntu-virtual-machine with the system setup to be accessible from the local system (at http://127.0.0.1). The login for the virtual machine is:

- Username: ubuntu
- Password: reverse
EVALUATION

The evaluation of the system will include two distinct forms of evaluation: a usability evaluation of the administration-interface that will determine its usability by end-users and a conceptual and technical evaluation of the web service that will determine to what extent the created service fulfills the goals of the Web 2.0 and REST ideologies.

6.1 Usability evaluation

The usability evaluation consisted of two stages:

First participants were asked to perform the following list of tasks in the administration interface:

- Create a new user.
- Create a new group.
- Assign permissions to this group.
- Find a specific publication.
- Edit the publication by changing the owner to the new user.
- Search and delete a comment.

During these tasks the participants were observed and their reactions and proceedings monitored.

After performing these tasks, the users were asked to fill out an online survey about their experience with the system they just used (the survey is printed in section 8.4).

The survey contained a usability section that was based on the SUS: this way the usability assessment could be performed utilizing a scientifically proven usability test (Sauro 2011).
6.1.1 Conduction of survey

The survey was conducted with 13 participants\(^1\): according to Tullis and Stetson (2004, p. 6) this is a great enough number for the SUS to allow correct conclusions from the tests, whereas ten or less participants would only lead to a correct conclusion rate of less than 80%.

The participants were recruited from a wide range of different background areas, to guarantee that schools with no computer science department could still administer the system. However, all participants either already finished one degree or were currently studying at a university.

The distribution of finished degrees is shown in the following graphic:

![Degree of education](image)

**Figure 6.1: Participant’s degrees**

Most participants rated their own skill in the use of computers and the internet rather well, and thought their own experience with system administration as average or good (see Figure 8.2 and Figure 8.3).

More than 75% were familiar with peer review as a term and about 30% have already conducted a peer review.

\(^1\)The complete results of the survey are provided on the accompanying DVD.
An enlightening insight could be gained from the questions about whether or not people would mind other people commenting or even peer reviewing their publications online: over 75% were in favour of peer reviewing online, whereas only one participant was against that practice (see Figure 8.4).

The usability part of the survey is evaluated using the methodology proposed in Brooke (1996, p. 5): in a first step the scores for each participant are calculated by assigning values from 0 to 4 to each answer (see Brooke (1996, p. 5) for the concrete process).

After this step, the scores calculated are summed together for each user and multiplied by 2.5 to normalize them on a scale from 0 to 100, which leads to the following values²:

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
<th>SUM</th>
<th>NORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>40</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>30</td>
<td>75</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>39</td>
<td>97.5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>26</td>
<td>65</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>32</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>27</td>
<td>67.5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>35</td>
<td>87.5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>33</td>
<td>82.5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>29</td>
<td>72.5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>34</td>
<td>85</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>36</td>
<td>90</td>
</tr>
</tbody>
</table>

Table 6.1: SUS score calculation

Dividing the normalized score by the number of participants leads to an average score of 81.54.

²A histogramm of the data can be seen in Figure 8.5
According to Sauro (2011) this equals an usability rating only achieved by the the first percentile in the SUS test.

To enhance the reliability of this test, the confidence interval for an $\alpha$ of 5% is calculated:

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x}) = 110.977;$$

$$[\bar{X} - z(1 - \frac{\alpha}{2} \frac{\sigma}{\sqrt{n}}); \bar{X} + z(1 - \frac{\alpha}{2} \frac{\sigma}{\sqrt{n}})] = [75.17; 87.90];$$

### 6.2 Conceptual and technical evaluation

The evaluation of the concepts implemented in the web service will be divided into three sections.

The first will outline a short evaluation of the performance of the system, whereas the second one will determine the adherence of the implementation to the Web 2.0 technologies. The third section will highlight certain points of the REST interface with regard to RESTful principles.

#### 6.2.1 Evaluation of system performance

Apart from evaluating the administrative user interface, the main part of the implementation is the web service that is exposed via the REST interface.

As this service could not be evaluated in usability aspects, a short evaluation of its performance shall be provided: the two most important actions are retrieving data, and inserting data.

These two actions were evaluated while the web service was deployed on the infrastructure of http://gondor.io/.

To evaluate the performance of GET requests, 1000 requests were performed in batches of 100 requests. The evaluation of the response times lead to the following results:

---

³According to Bangor et al. (2008, p. 576) the SUS usability test follows a standard deviation. This was used as basis for the following calculations.
In another test the performance of the batch-insertion of data from \texttt{BibTeX}-files was evaluated, as this might be used to insert great swathes of data at once:

For this test a \texttt{BibTeX} file with 143 records was inserted multiple times via a \texttt{POST}-request, which lead to the following results:

**Figure 6.2:** Performance of system for \texttt{GET} requests.
6.2.2 Evaluating the Web 2.0 technologies

This section will try to highlight how the implemented program attempts to adhere to the principles of the Web 2.0 that were introduced in section 2.1.

**Web 2.0 concepts:**

Some concepts can hardly be applied to only one service as they are a product of the architecture of the Web. These include:

- The *Web as a platform*.
- The *harnessing of the long tail*.
- The *end of the software release cycle*.
- *Data on an epic scale*.

Due to this fact these points will not be evaluated.

The remaining principles will be discussed in the following section.
Harnessing collective intelligence: This concept is fulfilled in the service’s generous user model: it is possible for everyone to create a user account for the service and to add or delete votes and ratings for publications and comments. This way it is possible to obtain new reviews and estimations for publications than the classic paper-based peer-review-system allows.

Lightweight programming models: Instead of allowing hackability of the service, the data is provided via a RESTful web service that can be utilized by other systems to add, edit or delete data as needed, thus allowing developers to add value to the service.

Software above the level of a single device: By providing the possibility to return different representations depending on the request it would be possible to add representations specifically tailored to the needs of special platforms. This would require a possibility to identify the source of the request (e.g. a smartphone could be identified by its User-Agent header (Frederick and Lal 2010, p. 98)) and allow the creation suitable representation. This feature is currently not part of the service. Taking into account the architecture it should be possible to add the feature with a mediocre effort.

The service tries to fulfill the common requirements of a Web 2.0 service and apart from the single device level seems to succeed.

6.2.3 Evaluating the REST implementation

This section will evaluate the web service’s design against the principles of REST as introduced in subsection 4.5.1.

Consistency & Uniformity: Each resource of the web service provides a certain number of HTTP-verbs that can be used and will have the effects as described in REST-web-services: an overview on page 46 and Richardson and Ruby (2007, p. 117).

However, not all verbs were implemented - due to their less-used nature than the core verbs HEAD and OPTIONS were omitted.
Addressability: This refers to the fact that all resources exposed in a RESTful manner are accessible via a URIs that can be transferred and reused. The web service tries to provide an addressable set of URIs.

Statelessness: In its current implementation, login data of users is contained in sessions that are set and read via cookies. The RESTful way to perform user authentication would be on a per-request basis that would require the client to resent login information with each request: support for this kind of authentication is possible by using either DIGEST or BASIC authentication that is supported by HTTP (Richardson and Ruby 2007, p. 278).

In the newest version of the Django framework, support for using the integrated authentication middleware with this kind of authentication is provided and would allow the migration from the current state to a more RESTful approach (D. S. Foundation 2011a).

Multiple resource representations: The current implementation provides only XML representations of the stored data. As can be derived from the design in subsubsection 4.5.2.2 it is possible to add new representations to the web service.

6.3 Fulfilled requirements

This section will summarize, which requirements specified in Requirements Analysis on page 23, were fulfilled:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB01</td>
<td>Fulfilled</td>
<td>Provides a REST and administrative interface.</td>
</tr>
<tr>
<td>DB02</td>
<td>Fulfilled</td>
<td>Provides a REST and administrative interface.</td>
</tr>
<tr>
<td>DB03</td>
<td>Fulfilled</td>
<td>Via Inserter and administrative interface.</td>
</tr>
<tr>
<td>DB04</td>
<td>Fulfilled</td>
<td>Via Inserter</td>
</tr>
<tr>
<td>DB05</td>
<td>Fulfilled</td>
<td>BibTeX types receive special validation.</td>
</tr>
<tr>
<td>DB06</td>
<td>Partial</td>
<td>Outside of system scope.</td>
</tr>
<tr>
<td>UM01</td>
<td>Fulfilled</td>
<td>Via REST and administrative interface.</td>
</tr>
<tr>
<td>Code</td>
<td>Category</td>
<td>Status</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>UM02</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td>UM03</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UM04</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UM05</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UM06</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UM07</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UM08</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UM09</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>WE01</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>WE02</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UI01</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td>UI02</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UI03</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UI04</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UI05</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UI06</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UI07</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UI08</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td>EI01</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>EI02</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td>EI03</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>EI04</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>EI05</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>UA01</td>
<td>Fulfilled</td>
<td>✔️</td>
</tr>
<tr>
<td>SE01</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td>SA01</td>
<td>Partial</td>
<td></td>
</tr>
</tbody>
</table>
Fulfilled – In administrative interface. REST-interface provides HTTP-status-codes.

Fulfilled – Data is accessible to choose referees. Passwords are stored securely.

6.4 Possible improvements

While implementing the system there often were points in the later stages, when it became obvious that there is a better approach to solve a problem that would however not be realisable in the given time frame.

These possible improvements will be summarized in the following paragraphs:

HTTP-authentication: As has been pointed out in subsection 6.2.3 the current system uses sessions and not the HTTP-authentication methods.

According to D. S. Foundation (2011a) it is possible to build the authentication system with HTTP-authentication, which would lead to a more RESTful service.

Dedicated search-framework: The current search (described in Search on page 62) is an ad-hoc implementation that tries to provide a basic search for most resources.

However, there exist a number of dedicated search-frameworks that can be integrated into Django and would provide full-text search and better scalability: Haystack⁴ can integrate different search backends (e.g. Xapian⁵ or Solr⁶).

Django-Sphinx⁷ provides full-text search without any special back ends.

Caching: The current application does not implement any special caching mechanism.

⁴http://haystacksearch.org/
⁵http://xapian.org/
⁶http://lucene.apache.org/solr/
⁷https://github.com/dcramer/django-sphinx
However, Django provides a caching-framework⁸ that can be integrated with different caching backends (e.g. Memcached⁹).

**HTML templates:** To allow the direct access of the core from the web, the implementation of HTML templates would provide access to all users that are familiar with using a web-browser.

---

⁸https://docs.djangoproject.com/en/1.3/topics/cache/
⁹http://memcached.org/
CONCLUSION

The main purpose of this work was to develop the core of a Web 2.0 based peer-review and publishing system that could be used to help in relieving the current system from the immense amount of work that it is exposed to.

Therefore, the thesis introduced the principles the Web 2.0 is based on and explained the basics of the currently established publication and peer-review processes. Moreover, it was not the first system with the intention to restructure the peer review process. The currently used systems that attempt to solve the same problem were introduced and compared to our system and their lack of real Web 2.0 features like tagging or rating were discovered and actively implemented in the new system.

Based on this comparison and the introduced principles, the system was built with a strong focus on a clean database design and the goal to provide a RESTful interface, to create an easily accessible and consumable web service that adheres to the known HTTP-standards. This was especially important to allow the possibility to expose the system’s Web 2.0 feature set to a wide variety of systems and thus a great user base. Moreover, it allows the system to make use of the Web’s advantages like scalability, loose coupling and statelessness: all goals that would not have been easy to fulfill would the system be based on a SOAP web service.

Due to the fact that the system itself does not provide a direct interface apart from the REST web service, an administration interface was implemented to allow the management of data via a web interface.

The implementation of a HTML representation for the current service should be one of the main foci of extensions that can be made to the system: to allow these extensions to be created in an easy fashion one of the main goals in the system’s was extensibility.

All these features were implemented with the help of the Python programming language and the Django web-framework.

Even though Django is a huge framework, the documentation provided on its website proved to be invaluable and up-to-date and helped a lot in implementing a RESTful web service to make the system extensible for future research.

However, due to the fact that the author of the system was largely unfamiliar
with Django, a lot of approaches that might have lessened the amount of code or that would have increased the usability of the system were only discovered in the end and proved to be too time-intense to implement: especially caching, HTTP-authentication and a better search framework would enhance the usefulness of the core platform.

Still, the framework and Python proved invaluably useful and without those technologies it would not have been possible to implement the system in its current form.

It is thus based, on the shoulders of giants.
APPENDIX

8.1 Tables
Table 8.1: BibTeX data fields (Part 1)
<table>
<thead>
<tr>
<th>Publication</th>
<th>Journal</th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article</td>
<td></td>
<td>Month</td>
</tr>
<tr>
<td>Book</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Booklet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inbook</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incollection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inproceedings</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manual</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Masterthesis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Misc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phdthesis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proceedings</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Techreport</td>
<td>Institution</td>
<td></td>
</tr>
<tr>
<td>Unpublished</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8.2: BibTeX data fields (Part 2)
Table 8.3: BibTeX data fields (Part 3)
8.2 Figures

**Overall computer usage**

- Very Bad (1): 7.69%
- Rather Bad (2): 7.69%
- Average (3): 46.15%
- Rather Good (4): 38.46%
- Very Good (5): 30.77%
- Don't Know (0): 7.69%

Figure 8.2: Participant's computer knowledge

**System administration**

- Very Bad (1): 15.38%
- Rather Bad (2): 7.69%
- Average (3): 38.46%
- Rather Good (4): 30.77%
- Very Good (5): 7.69%
- Don't Know (0): 7.69%

Figure 8.3: Participant's administration skills
Figure 8.1: CiteSeerX database schema
Would people allow their publications to:

- Be rated online
- Be commented on online
- Be Peer reviewed online

<table>
<thead>
<tr>
<th>Rating</th>
<th>No (1)</th>
<th>Rather No (2)</th>
<th>Don’t Mind (3)</th>
<th>Rather Yes (4)</th>
<th>Yes (5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall usage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.4: Participant’s on online rating/commenting/peer reviewing

Histogramm of SUS data.

Figure 8.5: Histogramm of the SUS data.
## Site administration

<table>
<thead>
<tr>
<th>Category</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groups</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Users</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Core Web Service</td>
<td></td>
</tr>
<tr>
<td>Authors</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Comments</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Esteems</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Further fields</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Keywords</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Paper groups</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Peer review templates</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Peer reviews</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Profile fields</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Publications</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Ratings</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Reference materials</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Research areas</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Tags</td>
<td>Add, Change</td>
</tr>
<tr>
<td>Votes</td>
<td>Add, Change</td>
</tr>
</tbody>
</table>

Figure 8.6: Front page of the administration interface.
Figure 8.7: Admin interface to add a publication.
Figure 8.8: Admin interface to modify a user.
8.3 Listings

8.3.1 Templates

Listing 8.1: gfx/ws/core_web_service/templates/xml/author.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<author xmlns="http://{{ url }}"
    xmlns:atom="http://www.w3.org/2005/atom">

    {{ author.name }}

    {{ author.address }}

    {{ author.affiliation }}

    {{ author.email }}

</author>
```

Listing 8.2: gfx/ws/core_web_service/templates/xml/authors.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<authors xmlns="http://{{ url }}"
    xmlns:atom="http://www.w3.org/2005/atom">

{% for author in author_list %}

    <author>

        <atom:link rel="author" type="application/xml"
            href="http://{{ url }}/author/{{ author.id }}"/>

    </author>

{% endfor %}

</authors>
```

Listing 8.3: gfx/ws/core_web_service/templates/xml/comment.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<comment xmlns="http://{{ url }}"
    xmlns:atom="http://www.w3.org/2005/atom">

    <title>

</title>
```

CHAPTER 8. APPENDIX

5    {{ comment.title }}
6 </title>
7 <text>
8    {{ comment.text }}
9 </text>
10 <date>
11    {{ comment.date }}
12 </date>
13 <publication>
14    <atom:link rel="publication" type="application/xml"
               href="http://{{ url }}/publication/{{ comment.publication.id }}"/>
15 </publication>
16 <user>
17    <atom:link rel="user" type="application/xml" href="
               http://{{ url }}/user/{{ comment.user.id }}"/>
18 </user>
19 <votes>
20 {% for vote in comment.vote_set.all %}
21    {% if not vote.votetype == 2 %}
22        <vote>
23            <atom:link rel="vote" type="application/xml"
                          href="http://{{ url }}/vote/{{ vote.id }}"/>
24        </vote>
25    {% endif %}
26    {% endfor %}
27 </votes>
28 </comment>

Listing 8.3: gfx/ws/core_web_service/templates/xml/comment.xml

1 <?xml version="1.0" encoding="utf-8"?>
2 <comments xmlns="http://{{ url }}"
3     xmlns:atom="http://www.w3.org/2005/atom">
4    {% for comment in comments %}
5        <comment>
6            <atom:link rel="comment" type="application/xml" href="
6                http://{{ url }}/comment/{{ comment.id }}"/>
7        </comment>
8    {% endfor %}
CHAPTER 8. APPENDIX

Listing 8.4: gfx/ws/core_web_service/templates/xml/comments.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<esteem xmlns="http://{{ url }}"
xmlns:atom="http://www.w3.org/2005/atom">
  <user>
    <atom:link rel="user" type="application/xml" href="http://{{ url }}/user/{{ esteem.userprofile.user.id }}"/>
  </user>
  <value>{{ esteem.value }}</value>
</esteem>
```

Listing 8.5: gfx/ws/core_web_service/templates/xml/esteem.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
{% if keywords %}
<keywords xmlns="http://{{ url }}"
xmlns:atom="http://www.w3.org/2005/atom">
  {% for key in keywords %}
  <keyword>
    <atom:link rel="keyword" type="application/xml" href="http://{{ url }}/keyword/{{ key.id }}"/>{{ key.keyword }}</keyword>
  {% endfor %}
{% else %}
  <keyword xmlns="http://{{ url }}"
xmlns:atom="http://www.w3.org/2005/atom">
    {{ keyword.keyword }}</keyword>
  {% endif %}
```

Listing 8.6: gfx/ws/core_web_service/templates/xml/keyword.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
```
Listing 8.7: gfx/ws/core_web_service/templates/xml/overview.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<papergroup xmlns="http://{{ url }}"
    xmlns:atom="http://www.w3.org/2005/atom">
    <title>{{ papergroup.title }}</title>
    <description>{{ papergroup.description }}</description>
    <blind_review>{{ papergroup.blind_review }}</blind_review>
    <editors>
        {% for editor in papergroup.editors.all() %}
        <editor>
            <atom:link rel="user" type="application/xml" href="http://{{ url }}/user/{{ editor.id }}"/>
        </editor>
        {% endfor %}
    </editors>
    <referees>
        {% for referee in papergroup.referees.all() %}
        % {
    </referees>
</papergroup>
</services>
```

Listing 8.7: gfx/ws/core_web_service/templates/xml/overview.xml
<referee>
  <atom:link rel="user" type="application/xml" href="
    http://{{ url }}/user/{{ referee.id }}" />
</referee>

{% endfor %}
</referees>

<tags>
{% for tag in papergroup.tags.all() %}
  <tag>
    <atom:link rel="user" type="application/xml" href="
      http://{{ url }}/tag/{{ tag.id }}" />
  </tag>
{% endfor %}
</tags>

<publications>
{% for publication in papergroup.publications.all() %}
  <publication>
    <atom:link rel="user" type="application/xml" href="
      http://{{ url }}/publication/{{ publication.id }}" />
  </publication>
{% endfor %}
</publications>
</papergroup>

Listing 8.8: gfx/ws/core_web_service/templates/xml/papergroup.xml

<?xml version="1.0" encoding="utf-8"?>
<papergroups xmlns="http://{{ url }}"
  xmlns:atom="http://www.w3.org/2005/atom">
{% for papergroup in papergroups %}
  <papergroup>
    <atom:link rel="papergroup" type="application/xml" href="
      http://{{ url }}/papergroup/{{ papergroup.id }}" />
  </papergroup>
{% endfor %}
</papergroups>

Listing 8.9: gfx/ws/core_web_service/templates/xml/papergroups.xml
CHAPTER 8. APPENDIX

Listing 8.10: gfx/ws/core_web_service/templates/xml/peerreview.xml

Listing 8.11: gfx/ws/core_web_service/templates/xml/peerreviews.xml
CHAPTER 8. APPENDIX

Listing 8.12: gfx/ws/core_web_service/templates/xml/peerreviewtemplate.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<peerreviewtemplate xmlns="http://{{ url }}"
   xmlns:atom="http://www.w3.org/2005/atom">
   <templatetext>
      {{ template.template_text }}
   </templatetext>
   <binarypath>
      {{ template.template_binary_path }}
   </binarypath>
</peerreviewtemplate>
```

Listing 8.13: gfx/ws/core_web_service/templates/xml/peerreviewtemplates.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<peerreviewtemplates xmlns="http://{{ url }}"
   xmlns:atom="http://www.w3.org/2005/atom">
   {% for peerreviewtemplate in peerreviewtemplates %}
      <peerreviewtemplate>
         <atom:link rel="peerreviewtemplate" type="application/xml"
           href="http://{{ url }}/peerreviewtemplate/{{ peerreviewtemplate.id }}"/>
      </peerreviewtemplate>
   {% endfor %}
</peerreviewtemplates>
```

Listing 8.13: gfx/ws/core_web_service/templates/xml/peerreviewtemplates.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<publication xmlns="http://{{ url }}"
   xmlns:atom="http://www.w3.org/2005/atom">
   <abstract>
      {{ publication.abstract }}
   </abstract>
   <address>
      {{ publication.address }}
   </address>
   <booktitle>
      {{ publication.booktitle }}
   </booktitle>
   <chapter>
      {{ publication.chapter }}
   </chapter>
</publication>
```
CHAPTER 8. APPENDIX

</chapter>
<doi>
{{ publication.doi }}
</doi>
<edition>
{{ publication.edition }}
</edition>
<editor>
{{ publication.editor }}
</editor>
<howpublished>
{{ publication.howpublished }}
</howpublished>
<institution>
{{ publication.institution }}
</institution>
<isbn>
{{ publication.isbn }}
</isbn>
<journal>
{{ publication.journal }}
</journal>
<number>
{{ publication.number }}
</number>
<organization>
{{ publication.organization }}
</organization>
<pages>
{{ publication.pages }}
</pages>
<publisher>
{{ publication.publisher }}
</publisher>
<review_status>
{{ publication.review_status }}
</review_status>
<series>
{{ publication.series }}
</series>
% Publication is under review 
{% if publication.review_status == 3 %}
  {% for pg in publication.papergroup_set.all %}
    # The current group is not blind reviewed 
    {% if not pg.blind_review %}
      # The user is a referee or an editor and may see the authors 
      {% if user in pg.referees.all or user in pg.

editors.all %}

<authors>

{% for author in publication.authors.all %}

<author>

<atom:link rel="author" type="application/xml"
href="http://{{ url }}/author/{{ author.id }}" />

</author>

{% endfor %}

</authors>

{% endif %}

{% endif %}

{% endfor %}

{% else %}

<authors>

{% for author in publication.authors.all %}

<author>

<atom:link rel="author" type="application/xml"
href="http://{{ url }}/author/{{ author.id }}" />

</author>

{% endfor %}

</authors>

{% endif %}

<comments>

{% for comment in publication.comment_set.all %}

<comment>

<atom:link rel="comment" type="application/xml"
href="http://{{ url }}/comment/{{ comment.id }}" />

</comment>

{% endfor %}

</comments>

<tags>

{% for tag in publication.tags.all %}

<tag>

<atom:link rel="tag" type="application/xml" href="
http://{{ url }}/tag/{{ tag.id }}" ></atom:link>

</tag>
{% endfor %}
</tags>

<keywords>
{% for keyword in publication.keywords.all %}
<keyword>
<atom:link rel="keyword" type="application/xml"
  href="http://{{ url }}/keyword/{{ keyword.id }}"></atom:link>
</keyword>
{% endfor %}
</keywords>

<referencematerials>
{% for rm in publication.referencematerial_set.all %}
<referencematerial>
<atom:link rel="referencematerial" type="application/xml"
  href="http://{{ url }}/referencematerial/{{ rm.id }}"></atom:link>
</referencematerial>
{% endfor %}
</referencematerials>

<fields>
{% for field in publication.furtherfield.all %}
    <{{ field.key }}>{{ field.value }}</{{ field.key }}>
{% endfor %}
</fields>
{% if publication.decorated %}
<decorated>
{% for key, value in publication.decorated.items %}
    <{{ key }}>{% first %}{% endfirst %}</{{ key }}>
{% endfor %}
</decorated>
{% endif %}
{% endif %}
Listing 8.14: gfx/ws/core_web_service/templates/xml/publication.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<publications xmlns="http://{{ url }}"
    xmlns:atom="http://www.w3.org/2005/atom">
{% for publication in publication_list %}
    <publication>
        <atom:link rel="publication" type="application/xml"
            href="http://{{ url }}/publication/{{ publication.id }}" />
    </publication>
{% endfor %}
</publications>
```

Listing 8.15: gfx/ws/core_web_service/templates/xml/publications.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<rating xmlns="http://{{ url }}"
    xmlns:atom="http://www.w3.org/2005/atom">
{% if ratings %}
    {% for rating in ratings %}
        <rating>
            <atom:link rel="rating" type="application/xml"
                href="http://{{ url }}/rating/{{ rating.id }}" />
        </rating>
    {% endfor %}
{% else %}{% if rating %}
        <publication>
            <atom:link rel="publication" type="application/xml"
                href="http://{{ url }}/publication/{{ rating.publication.id }}" />
            <rating>{{ rating.rating }}</rating>
        </publication>
    {% endif %}{% endif %}
</rating>
```

Listing 8.16: gfx/ws/core_web_service/templates/xml/rating.xml
CHAPTER 8. APPENDIX

1 <?xml version="1.0" encoding="utf-8"?>
2 <ratings xmlns="http://{{url}}" xmlns:atom="http://www.w3.org/2005/atom">
3 {%- for rating in ratings %}
4   <rating>
5     <atom:link rel="rating" type="application/xml" href="http://{{url}}/rating/{{rating.id}}"/>
6   </rating>
7 {%- endfor %}
8 </ratings>

Listing 8.17: gfx/ws/core_web_service/templates/xml/ratings.xml

1 <?xml version="1.0" encoding="utf-8"?>
2 <referencematerial xmlns="http://{{url}}" xmlns:atom="http://www.w3.org/2005/atom">
3   <publication>
4     <atom:link rel="publication" type="application/xml" href="http://{{url}}/publication/{{referencematerial.publication.id}}"/>
5   </publication>
6   {%- for referencematerial in referencematerials %}
7     <referencematerial>
8       <atom:link rel="referencematerial" type="application/xml" href="http://{{url}}/referencematerial/{{referencematerial.id}}"/>
9     </referencematerial>
10   {%- endfor %}
11 </referencematerials>

Listing 8.18: gfx/ws/core_web_service/templates/xml/referencematerial.xml
Listing 8.19: gfx/ws/core_web_service/templates/xml/referencematerials.xml

```xml
<researcharea xmlns="http://{{ url }}" xmlns:atom="http://www.w3.org/2005/atom">
  {{ researcharea.title }}
</researcharea>
```

Listing 8.20: gfx/ws/core_web_service/templates/xml/researcharea.xml

```xml
<researchareas xmlns="http://{{ url }}" xmlns:atom="http://www.w3.org/2005/atom">
{% for researcharea in researchareas %}
  <researcharea>
    <atom:link rel="researcharea" type="application/xml" href="http://{{ url }}/researcharea/{{ researcharea.id }}"/>
  </researcharea>
{% endfor %}
</researchareas>
```

Listing 8.21: gfx/ws/core_web_service/templates/xml/researchareas.xml

```xml
<tag xmlns="http://{{ url }}" xmlns:atom="http://www.w3.org/2005/atom">
  {{ tag.name }}
</tag>
```
CHAPTER 8. APPENDIX

Listing 8.22: gfx/ws/core_web_service/templates/xml/tag.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<tags xmlns="http://{{ url }}"
xmlns:atom="http://www.w3.org/2005/atom">
{% for tag in tags %}
  <tag>
    <atom:link rel="tag" type="application/xml" href="http://{{ url }}//{{ tag.id }}"/>
  </tag>
{% endfor %}
</tags>
```

Listing 8.23: gfx/ws/core_web_service/templates/xml/tags.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<quser xmlns="http://{{ url }}"
xmlns:atom="http://www.w3.org/2005/atom">
  <username>{{ quser.username }}</username>
  <first_name>{{ quser.first_name }}</first_name>
  <last_name>{{ quser.last_name }}</last_name>
  <email>{{ quser.email }}</email>
  <degree>{{ quser.profile.degree }}</degree>
  <institution>{{ quser.profile.institution }}</institution>
  <authenticated_professional>{{ quser.profile.authenticated_professional }}</authenticated_professional>
</quser>
```
CHAPTER 8. APPENDIX

Listing 8.24: gfx/ws/core_web_service/templates/xml/user.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<users xmlns="http://{url}"
xmlns:atom="http://www.w3.org/2005/atom">
{% for user in users %}
  <user>
    <atom:link rel="user" type="application/xml" href="http://{url}/user/{{ user.id }}"/>
  </user>
{% endfor %}
</users>
```

Listing 8.25: gfx/ws/core_web_service/templates/xml/users.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<vote xmlns="http://{url}"/>
```
CHAPTER 8. APPENDIX

Listing 8.26: gfx/ws/core_web_service/templates/xml/vote.xml

8.3.2 View code

8.3.3 Business logic code

Listing 8.27: BibTeX parser BNF
This survey will help evaluate the usability of the admin interface that is part of the core-knowledge-platform that was developed as part of a master thesis at the Heriot-Watt University.

Welcome

Thank you for taking the time to answer the following survey! Your feedback is very important to us, as it will help us in improving the system.

This survey should only take about 10 minutes of your time. Your answers will be completely anonymous.

In order to progress through this survey, please use the following navigation buttons:

* Click the Next button to continue to the next page.
* Click the Previous button to return to the previous page.
* Click the Exit and clear survey button if you want to exit the survey.
* Click the Resume later button if you want to finish the survey at a later stage.
* Click the Submit button to submit your survey.

There are 7 questions in this survey

Technical and academic background

These answers will give us some information about your technical and academic background.

1 [bi1] What is your highest degree of education?

Please choose only one of the following:

- Bachelor's degree (or equivalent)
- Master's degree (or equivalent)
- Doctor's degree
- High School
- Other

If your degree was obtained in a foreign language and you know the English translation please provide it if possible.

Figure 8.9: Survey (page 1)
3 [bi2]How would you describe your skill level in the following areas: *

Please choose the appropriate response for each item:

<table>
<thead>
<tr>
<th></th>
<th>Very Bad</th>
<th>Rather Bad</th>
<th>Average</th>
<th>Rather Good</th>
<th>Very Good</th>
<th>Don't Know</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall computer usage</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Internet usage</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>System administration</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

4 [bi3]Are you familiar with the term "peer review"? *

Please choose only one of the following:

- ○ Yes
- ○ No

5 [bi4]Have you performed a peer review yourself already? *

Only answer this question if the following conditions are met:
* Answer was 'Yes' at question '4 [bi3] (Are you familiar with the term "peer review")

Please choose only one of the following:

- ○ Yes
- ○ No

6 [bi5]If you publish a paper (book, thesis or other text): *

Please choose the appropriate response for each item:

<table>
<thead>
<tr>
<th></th>
<th>No</th>
<th>Rather No</th>
<th>Don't Mind</th>
<th>Rather Yes</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Would you let people rate it online?</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Would you let people comment on it online?</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Would you let it be peer-reviewed online?</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

Figure 8.10: Survey (page 2)
ystem Usability Scale

This question group will evaluate the usability of the administration according to the system usability scale.

<table>
<thead>
<tr>
<th></th>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neither agree nor disagree</th>
<th>Agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>I think that I would like to use this system frequently.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>I found the system unnecessarily complex.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>I thought the system was easy to use.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>I think that I would need the support of a technical person to be able to use this system.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>I found the various functions in this system were well integrated.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>I thought there was too much inconsistency in this system.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>I would imagine that most people would learn to use this system very quickly.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>I found the system very cumbersome to use.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>I felt very confident using the system.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>I needed to learn a lot of things before I could get going with this system.</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Figure 8.11: Survey (page 3)
BIBLIOGRAPHY


— (2011c). Django 1.0 release notes. Django Software Foundation. URL: https://docs.djangoproject.com/en/1.3/releases/1.0/.


Patașnik, Oren (Feb. 1988). *BibTeXing*.

Rowland, Fytton (2002). “The Peer review Process A Report to the JISC Scholarly Communications Group Loughborough University”. In:
“InfoQ Explores REST” (Mar. 2010). In: ed. by Ryan Slobojan.


DECLARATION

I, Florian Bergmann,

confirm that this work submitted for assessment is my own and is expressed in my words. Any uses made within it of the works of other authors in any form e.g. ideas, equations, figures, text, tables, programs are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: ..............................

Date: ..............................