

**Retina-Destruction:
Designing, Developing and Securing a Web Application
Chantzis Dimitrios
Master of Science (Design and Digital Media)
School of Arts, Culture and Environment
The University of Edinburgh
2010**



**Retina-Destruction:
Designing, Developing and Securing a Web Application
Chantzis Dimitrios
Master of Science (Design and Digital Media)
School of Arts, Culture and Environment
The University of Edinburgh
2010**



Acknowledgments

I would like to thank my family for all their trust and support, my friends in the Design and Digital Media Program and Karolina Diamanti for her infinite patience, help, support and cooking lessons during this academic year.

Table of Contents:

Submission Details – p.9
0: Abstract – p.10
1. Introduction – p.11
2. System Functions Showcase – p.14
3. Interface Design – p.20
4. The Database – p.26
5. Code Authoring Methodologies & Files Organization – p.33
6. System Security – p.37
Appendices – p.43
References – p.74
Bibliography – p.74
Video Tutorials – p.79
Code Libraries – p.79
Useful Websites – p.80

Appendices:

Appendix A: DVD contents – p.43
Appendix B: Installation Guide – p.44
Appendix C: Database & Database Users SQL Code – p.48
Appendix D: Database Tables & Demonstration Entries – p.56
Appendix E: Video Files Contents – p.69
Appendix F: System Generated E-Mail Messages – p.70

List of Figures:

2.1: The central interface. - p.15
2.2: The Account Interface.- p.18
2.3: A generated portfolio website. - p.19
3.1: The database tables. - p.21
3.2: Privileges of user 'user1' for all the database tables. - p.24
3.3: Privileges of user 'user2' for all the database tables. - p.25
4.1: Jakob Nielsen's, “Ten Usability Heuristics”(2005). - p.26

- 4.2: <http://www.magne-f.net/1/> - p.28
- 4.3: A selected image presented on a new layer. - p.29
- 4.4: The design editor. - p.31
- D.1: Structure of database table 'album'. - p.56
- D.2: Demonstration entry for database table 'album'. - p.56
- D.3: Structure of database table 'comment'. - p.57
- D.4: Demonstration entry for database table 'comment'. - p.58
- D.5: Structure of database table 'fileformat'. - p.58
- D.6: Demonstration entry for database table 'fileformat'. - p.58
- D.7: Structure of database table 'image'. - p.59
- D.8: Demonstration entry for database table 'image'. - p.60
- D.9: Structure of database table 'posts'. - p.60
- D.10: Demonstration entry for database table 'posts'. - p.61
- D.11: Structure of database table 'tag'. - p.62
- D.12: Demonstration entry for database table 'tag'. - p.62
- D.13: Structure of database table 'template'. - p.62
- D.14: Demonstration entry for database table 'template'. - p.63
- D.15: Structure of database table 'user'. - p.63
- D.16: Demonstration entry for database table 'user'. - p.65
- D.17: Structure of database table 'usersactionlog'. - p.67
- D.18: Demonstration entry for database table 'usersactionlog'. - p.67
- D.19: Structure of database table 'website'. - p.66
- D.20: Demonstration entry for database table 'website'. - p.68
- F.1 Account activation e-mail for newly registered users. - p.70
- F.2: Password reset e-mail, for users that filled the “forgot your password?” form. - p.70
- F.3: E-mail to the default system account, mal@retinadestruction.com using the system's contact form. - p.71
- F.4: E-mail to a user who sent a message to the default system, mail@retinadestruction.com using the system's contact form and selecting “Send CC to self“. - p.71
- F.5 Notification e-mail for new comment that was left regarding an image album of a registered user. - p.72
- F.6: Notification e-mail for new comment that was left regarding an blog entry of a registered user. - p.72
- F.7: E-mail to a registered user of the system. This message was sent by a visitor filling the contact form from the registered user's portfolio website (screenshot from Gmail). - p.73

Submission Details

This submission consists of two parts. A complete functioning web application, which is the most important part of the project, and this publication, which contains all the necessary information about understanding the system and its functions. It also includes my research and implementation on the subject matters of interface design, system design and security for the submitted web application.

The source code of the web application is included in the accompanying DVD, and is available at

<http://dimitrioschantzis.com/rdsystem/>

Full instructions on how to install the system on a local server are included in Appendix B.

Another version of this application is available at

<http://retinadestruction.com/>

and uses a database of a different name. This version should not be used for the review of the project.

I have also included videos that demonstrate almost all operations that the system offers. The contents of each video is listed in Appendix E.

Lastly, the contents of the accompanying DVD is presented in Appendix A.

0 . Abstract

The objective of this dissertation is the design and implementation of a web application called Retina-Destruction. This system is used to generate customizable portfolio websites for visual artists. The contents and interface design of these websites are completely customizable by the registered users. These websites are hosted in the system and are accessible from a unique URL address. Emphasis is given to the ways of securing web applications and their implementation to this system through carefully designed algorithms as well as the instructions provided by the OWASP (Open Web Application Security Project) community.

The written part of this project analyzes the functions of the system, the structure of the database, the logic behind the organization of the code functions in files and presents some selected operations of the system. Also included is a breakdown of the graphical interfaces of the system, and the decisions behind them.

Used Technologies:

XHTML, PHP, CSS, JavaScript, AJAX (jQuery library), MySQL, JSON.

The system is temporarily accessible at: <http://dimitrioschantzis.com/rdsystem/>.

There is a testing account with **username:** james.doe@gmail.com and **password:** *qwaszxqwaszx*

1. Introduction

This paper intends to provide the reader with all the necessary information to understand the functions of the web application called Retina-Destruction and the decisions behind its production. The development of this system is the main part of my final project.

It is advised for the reader to have a fairly good understanding of the digital technologies behind the production of interactive web applications. Nonetheless, even though the nature of this dissertation subject demands the use of technical terms and some algorithmic methodologies, I tried to limit them to just two chapters.

Retina-Destruction is an online platform specifically designed to empower visual artists with a complete online presence. First and foremost, this system is a tool to be used for generating completely customizable portfolio websites, in terms of content and interface design, primarily for visual artists. The generated websites are hosted in the system under a unique URL.

Due to the time limitations of this project, some options are limited, even though they were designed and the necessary infrastructure was implemented. The description of these functions is included in the first chapter.

This online platform was built using XHTML, CSS, JavaScript, the JQuery AJAX library, PHP, JSON and a MySQL relational database.

The development of a web application using these technologies can vary in difficulty, according to the system in question. However:

- the right organization and the production of a canonical database that will not store redundant and irrelevant data,
- the design of a graphical interface for the application, that will provide its users with the most achievable usability and not demand detailed instructions for every function it performs,
- the authoring of appropriate algorithms for the system's functions that will not weigh down its performance and
- implementing security measures that will maintain the integrity of the database and the right function of the system

demand a very careful design, discipline in the manifestation and a clear approach to all the functions that the system has to execute.

This was the first web application that I had built using the JQuery ALAX library. Up until that point, I had been using the mootools one. I decided that the production of this project would be a good chance to learn another library, which seems to be a lot more popular ^{1,2}. As a result, I re-evaluated and re-wrote almost all

the functions that I had designed and used in all my previous systems.

As it is most common with the web applications that I create, Retina-Destruction is a work in progress, constantly changing and transforming. All the code functions, as well as the database, were designed to be flexible to changes and expansions. Whenever I had a function finished and working efficiently, a few days later a different and more advanced algorithm would manifest and linger in my mind, until it was incorporated into its previous version. The version that is presented and analyzed in this paper is the one that I am currently most satisfied with.

1.1 Conception

I should mention that one of my greatest passions is the fine arts. From sketches and scribbles, to full illustrations and paintings, using traditional “analogue” media. In order to get myself motivated for producing and digitally publishing work of more professional magnitude, I applied my background in information technologies and design to create a portfolio website that incorporated a specifically implemented content managing system. This website is currently in its third version and available at www.jamesdoe.com.

My initial idea for Retina-Destruction came about 3 years ago, while I was designing the third version of my portfolio website. I realized that with only some minor changes to the database and the code functions, the system would be usable by more than one user. I started taking notes on how this system that would serve a fair number of users, in a very simple and efficient way and without the user needing any technical knowledge. I already had some ideas about including a design editor for the graphical interfaces.

I was not satisfied with the algorithms I wrote for several code functions in terms of system security and performance. However, in order to create a new system of this magnitude, I needed time to research similar systems and new technologies, and a lot more experience in web design.

The target users of this system would be artists whose work can be represented **visually** in images or video. According to the Encyclopedia Britannica website³:

[Retina is a] layer of nervous tissue that covers the inside of the back two-thirds of the eyeball, in which stimulation by light occurs, initiating the sensation of vision.

And while researching some articles about art, I stumbled upon an article by Bobby Matherne⁴ (2002), stating that:

[...]the Oscar Wilde has it rightly; Art is the process of the destruction of sameness found in the material world when it first appears.

Thus, I decided to call this system “Retina-Destruction”.

While I was planning my final project proposal, I revisited my earlier notes and ideas about this application and began revising them in paper. My only concern was the time limitations of the project. Since the most important part of the MSc in Design and Digital media is research and experimentation, I decided to create a similar system which featured a very limited set of the functionality that I wanted to include in Retina-Destruction. This project was my submission for the “Dynamic Web Design” class of the course, and proved to me that the production of Retina-Destruction was achievable, even though it demanded a lot of hard work.

I extended my proposal to include research that I wanted to work on, regarding the interface and functionality design of web applications, as well as the guidelines and methodologies on their security. I also included some decisions I made for optimizing the performance of these systems.

2. System Functions Showcase

2.1 Introduction to the System

Retina-Destruction is an online web application that generates customizable websites, in terms of content and design. The system was designed to produce portfolio websites for visual artists, which would exist concurrently in the systems hosting server. However, it can be used to produce any other kind of websites with similar content. It consists of three main interfaces: the central home page, the registered users account interface and the generated websites. Detailed descriptions of these pages are presented in the contents of this chapter.

There are three different user types, each one with different access privileges to the functions of the system and the database tables controlled by them.

The **administrator** user who has full privileges in the all database tables controlled by the system. This user also has access to functions that can activate/deactivate the accounts and websites of the registered users, view the log file where all the system's errors are stored, and an action log containing some of the functions that the registered users access (sign-in, sign-out, etc.). Due to the very strict time limitations of this project, I decided not to implement any of the administrator's functions, and focused more on proof testing and debugging the rest of the system that I had completed.

The **common** users who can browse the public contents of the system such as the section containing the full list of registered artists with public accounts, as well as the registration, sign-in and contact forms. Common users can also browse all the public websites of the system.

The **registered** users who have the same privileges as the common, but can also access a sophisticated section where they can manage their account and edit the contents and interface design of their websites.

In this chapter I present all the functionality that is provided throughout the system. I chose to reference these features in a dedicated chapter as I believe that it is very important for the reader to understand exactly what the system has to offer. In chapter 4, titled "Interface Design" I analyze the reasons for including the following functionality for each page section.

The accompanying DVD contains a number of video files demonstrating the functions of the system. These files are in a folder named `"/VIDEO_DEMOS/".`

2.2 The Central Interface

This interface can also be called the “home page” of the system, as it is the first one presented to the users when the system is loaded. It includes seven main sections and two subsections.

1. **Introduction:** It contains some introductory information about the nature of the system and definitions of the terms “retina”⁵ and “destruction”⁶.
2. **Artists Index:** Contains a full list of registered artists that have a public profile. Included is a list of all the available categories that are available to filter the list.
3. **Featured Artists:** The five registered artists whose websites have been viewed the most.
4. **About:** General information about the system.
5. **Contact:** A form that can be used to send messages to the default e-mail account of the system.]
6. **Register:** Includes the registration form and a link to a subsection containing the terms & conditions of the system.
7. **Sign-In:** Includes the form used to log-in the system, and a link to a subsection which contains a form that can be used by the registered artists to reset their password, if they have forgotten it.

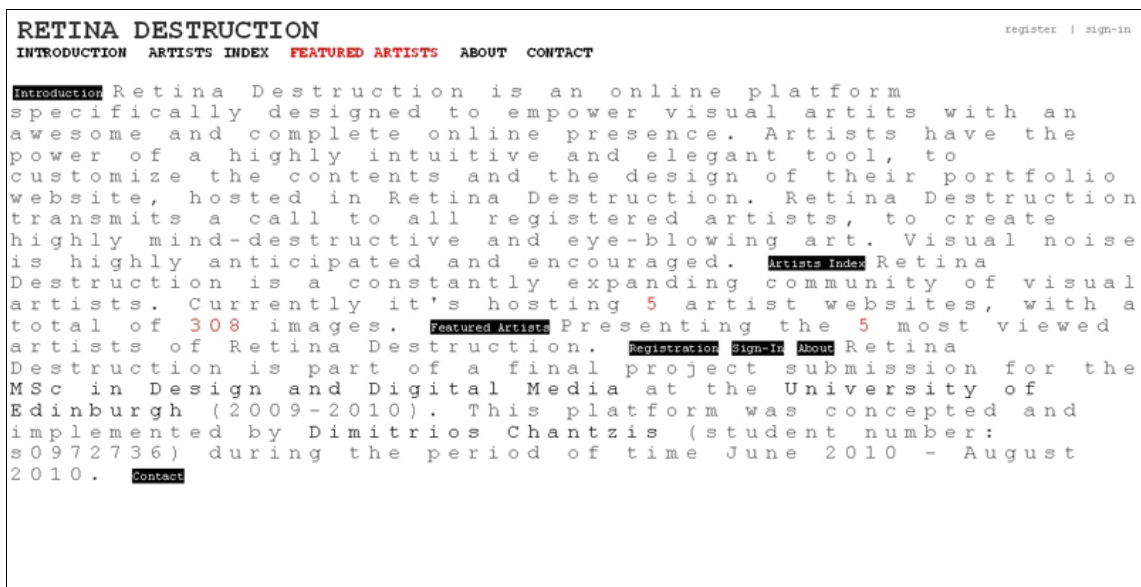


Figure 2.1: The central interface.

Whenever new users register to the system for a new account, a notification message containing an activation link is sent to the e-mail address they specified. If they try to sign-in without previously activating their account, the system will not recognize the submitted credentials.

There is another case where the system sends an activation link via e-mail. It is when users have forgotten

their password and cannot sign-in. This activation link redirects them to a page where they can reset their password. Examples of all the system-generated e-mails can be found in Appendix E.

In the first iteration of the system I added a section to the home page titled 'Instructions', but later on I chose to move it to the “My Account” interface as it seemed to be more helpful there.

A presentation of these sections can be found in the video files “**video01_main_website.mp4**” and “**video02_account_actions.mp4**”.

2.3 The Account Interface

This page is accessible only by the registered users after they sign in. It includes a list of sections from which users can edit their account information, manage their website contents and design.

1. **Overview:** This section gives a general view of each users' account, statistics regarding their profile and website. The information included there can be altered from the "settings" section.
2. **Settings:** A section collecting all the settings and options that help the users
 - edit their profile and website by adding a description,
 - select artist categories that describe their work,
 - add social media links and a profile image,
 - change the privacy options of their account,
 - reset their password and
 - define how their images are uploaded and resized for their website.
3. **Albums:** Provides the users with a full set of options about the following matters:
 - Create, update and delete albums
 - Add descriptions for each album and embed videos from sources such as youtube.com and vimeo.com
 - Album categorization. Albums that belong in the same category are grouped together. Albums not belonging in any category are presented as separate pages.
 - Upload covers for the albums
 - Edit the visibility of each album (invisible albums are not accessed by visitors)
 - View all the comments regarding each album and respond to them

- Select how the images are presented in the album from three available options: full size, thumbnails and large thumbnails. Every album can have a different option selected.
- Upload, re-order and delete album images
- Add captions and tag keywords for each image. Visitors can search all images under the same tag.
- Set the way your images are presented for each album

NOTE: For every image that is uploaded to a website, a new blog entry containing that image is automatically created. All images uploaded on the same date, are grouped together in the same blog entry. This is an automatic action. These blog entries are completely customizable and by default invisible to the website visitors.

4. **Blog:** Includes options that are very similar to the album options. Users can:

- Create, update and delete blog entries
- Add a message for each blog entry, and embed videos from sources such as youtube.com and vimeo.com
- Sort blog entries by tags (keywords). Visitors can search all blog entries under the same tag.
- View all the comments regarding each album and respond to them
- Select how the images are presented in a blog entry. Every entry can have a different option selected.
- Edit the visibility of each entry
- Upload, re-order and delete album images
- Add captions and tags for every image. Visitors can search all images under the same tag.
- Set the way your images are presented for each blog entry.

NOTE: Every image that is uploaded inside a blog entry is saved in a specific album that is created automatically. This album is by default invisible to the website visitors.

5. **Cover Page:** It is the page that first appears when the portfolio website is loaded. The users can choose if they want this home page to be 1) empty, 2) consist of their blog section, or 3) display a randomly selected image.

NOTE: Images that are uploaded to be used as random cover images are saved in a specific album that is created automatically. This album is by default invisible to the website visitors.

6. **Design Editor:** This is the most elaborate section of the account page. It contains all the necessary settings to change the "look and feel" of the generated websites. There are 4 templates which contain predefined values for all the design settings. From there users can change each one to fit their individual website presentation needs.

These settings include:

- The orientation of the website navigation bar (left, right or top)
- Color palettes for the different types of normal and highlighted text
- Uploading fields for wallpaper and logo images
- Various fonts
- Optional information to be included in the sidebar of the website such as an images tag-cloud, the user's e-mail address and the date the website was last edited.

7. **Instructions** about all the previous sections.



Figure 2.2: The Account Interface.

The video files “**video03_myAccount_operations_01.mp4**” and “**video03_myAccount_operations_02.mp4**” demonstrate the functions of this page. The design editor is presented in the last video file “**video03_myAccount_operations_01.mp4**”.

2.4 The Generated Website

Each website that the system generates combines the information that is submitted and uploaded from the previous interface. This information is organized in a very simple and intuitive way, in terms of navigation. The visitors of each website

- can browse through all the visible image albums and blog entries of the artist and leave their comments,
- navigate to the “about” page, which includes the account information of the user,
- send an e-mail to the artist from the contact form of the website and

- search for images that share the same tags. This feature is also available for blog entries.

The last video file “**video03_myAccount_operations_03.mp4**” displays a generated website.



Figure 2.3: A generated portfolio website.

3: The Database

3.1 Introduction

I always treat the database of a web application as the most important part of development. It demands very careful organization of the data it will store. A developer should keep in mind which information needs to be stored and how to acquire this information using as little requests to the database as possible.

The Retina-Destruction system uses a database that consists of ten tables. For web applications that require a user to register, but also provide content for the common visitors, I usually create three types of database users to establish the connections to the database, each one with a different set of privileges for the tables.

The Appendix B, contains all the information needed to install the database of the system.

In Appendix C I included all the necessary SQL code for creating the database, the tables and the users, but also detailed descriptions of the table fields and demonstration entries for all the tables.

3.2 The Database

All the information stored in the database is organized in 10 tables. Figure 3.1 gives an overview of these tables. Each table was designed 1) to include all the data the system needs to function properly and 2) to store the maximum possible information for each entry without having to store duplicate entries. Most importantly I wanted the data organized in such a way that it can be accessed with the minimum possible requests, thus avoiding unnecessary ones, in order keep the response time of the system's functions low.

In some occasions I felt that tables included redundant data that was also stored in others. However, reducing those would have resulted either in more complex queries to the database, or a larger number of queries.

Table Name	Fields Count
album	16
comment	8
fileformat	5
image	16
posts	14
tag	5
template	7
user	25
usersactionlog	8
website	20

Figure 3.1: The database tables.

An example for this are the tables 'album', 'comment', 'image' and 'posts'. The system offers all registered users multiple albums, images and blog entries. Each one of the images can belong to an album, a blog entry, or even both. These are images uploaded by the user. I also needed the images to be saved in a certain order that can easily be edited by the user.

One way of saving this information is by adding two fields: one for the place of this image in an album, and another one for its place in a blog entry. Every time a user decides to change this order in an album or blog entry of just one image, all the included image entries have to be updated with their new order number. Since there is no limit for included images¹, and if we keep in mind that the images' positioning is a drag-n-drop feature with the possibility of an unknown number updates where each one of them should take less than half a second, then we can imagine the case of a 150 images album, where the user decides to change the position of **every** single image!

Since this case goes against my philosophy of minimal database requests, I decided to store the same information in the 'album' and 'posts' tables. These fields have the following structure:

Table-ID-of-image-1:.....:Table-ID-of-image-2:.....:Table-ID-of-image-3:.....: ...

This way, only one field of one entry gets updated each time the positioning function is executed.

¹ Even if I added an upper limit to the number of images, that would be among 70 to 100 for each album, which is still a substantial number of queries to be executed all at once for a minor function like the images position reorder.

The same logic is applied in the following cases:

- Image tagging and blog entry tagging with keywords entered by the user. Each image and blog entry can have multiple tags assigned to them
- User profile categorization, where a user can characterize his profile by selecting multiple keywords

Another very interesting table structure that I would like to analyze here is the one of the 'website' table.

Every registered user can have only one website in the system. Among the most typical fields of this table (id number, title, views counter) I added several fields that would describe how this website is rendered:

- **wallpaper** and **logo**. If the user has uploaded the images, then these fields have the values “wallpaper.png” and “logo.png” accordingly, in any other case these fields have the **null** value².
- **sidebar**. The user can select to add to the sidebar of their website 1) the date that their website was last edited, 2) a copyright³ text for the images as well as 3) their e-mail address.

If the only the e-mail address is selected to appear in the sidebar this field would have the value:

updated:0:...:copyright:0:...:email:1

- **imagetagcloud**. Can either have the value “yes” or “no”. It specifies if an images tag cloud will appear in the sidebar of the website.
- **sidebarorientation**. The sidebar of each website can have a left, right or top positioning.
- **designcsscode**: This field is designed to store all the CSS code that is unique for each website. In the current version of the system this field has the **null** value, because all the generated websites share very similar templates. For now the “**designformoptions**” field is used to store all the design parameters. In a later version of the system, the CSS code stored here would be used to style the corresponding website.
- **designformoptions**. I added this field to store the design parameters that are edited by the user, and are used to produce the website CSS code. At first this would be used only to store the values, so that they can be loaded into the appropriate forms of the interface to show the users their selections. Later I realized that the CSS code for all the possible layouts is similar enough to get generated fast for each website and, at least for this version of the system, I did not use the '**designcsscode**' field.

The structure of the values saved in this field, can seem to be very complex:

2 **Update:** In my most recent tests, there were some bugs with the save process of this information, so these fields always have the values “wallpaper.png” and “logo.png” accordingly, even when the user does have these images uploaded. CSS is a forgiving language, and if it tries to add an image to the design of the page that does not exist, it will not generate an error.

3 I decided to temporarily remove the copyright selection from the interface forms, until I have researched the copyright subject a little more. The value of the copyright parameter is by default the zero value (0) and users are not able to update it.

wallpaper:image:.....wallpapercolor:ffffff:.....logo:text_image:.....sidebarorientation:right:.....:
 text:textcolor_000000:.....text:linkcolor_000000:.....text:linkbgcolor_:.....text:linkhovercolor_f
 ffff:.....text:linkhoverbgcolor_000000:.....text:navilinksize_small:.....sectionsbg:trans_white_li
 ght:.....text:fontfamily_Georgia, "Times New Roman", Times,
 serif:.....text:highlightcolor_ffffff:.....text:bghighlightcolor_000000:.....wallpapertype:image

A few words about each parameter:

- **wallpaper: image:** Defines that the website has a background image.
 - **wallpapercolor:ffffff:** The website has a white background color.All parameters that include a color value use a hexadecimal notation (HEX). If a color parameter is empty then there is no color defined.
 - **logo:text_image:** The logo of the website Is the image uploaded by the user. Other possible values are **text_username** and **text_websitename**
 - **sidebarorientation:right:** Sets the orientation of the sidebar.
 - **text:textcolor_000000:** Sets the color of normal text in the layout.
 - **text:linkcolor_000000:** Sets the color of text links.
 - **text:linkbgcolor_:** Sets the background color of text links
 - **text:linkhovercolor_:** Sets the color of text links when the mouse cursor hovers on them
 - **text:linkhoverbgcolor_000000:** Sets the background color of text links when the mouse cursor hovers on them.
 - **text:navilinksize_small:** Sets the size of the navigation text links. Possible values are **small**, **smallest**, **medium** and **large**.
 - **sectionsbg:trans_white_light:** Sets that the background of the website sidebar and main section. This can be a transparent white layer., a transparent black layer, a color or nothing. Each transparent layer has a light and a dark version for the user to select.
 - **text:fontfamily_Georgia, "Times New Roman", Times, serif:** Sets the font of the text. There are 5 different fonts supported by the design editor.
 - **text:highlightcolor_ffffff:** Some text in the design of the website needs to be highlighted. This parameter sets the color of that text.
 - **text:bghighlightcolor_000000:** Sets the background color of highlighted text.
 - **wallpapertype:image.** An additional parameter about the background image of the website.
- **coverpage.** The cover page of each website can either be a randomly selected image, the blog section of the website, or empty. This field stores the appropriate user selection.

The structure of this table can be optimized if the fields 'coverpage', 'sidebar', 'imagetagcloud' and

'sidebarorientation' were appended as parameters in the 'designformoptions' field. I added references of these fields in the code functions of the system prior to the addition of the 'designformoptions' field.

3.3 The Database Users

Whenever the code functions of a web application need to export (or import) data to a database, a connection must be established between the two them. This connection needs the credentials of a database user, with a specific set of privileges to access the data of each table.

For this application, in order to increase the protection of the database I added three database users:

- The 'admin' database user, who has full privileges to all the database tables and is only used when the system administrator signs into the system
- The 'user1' database user. This one is used when a registered user signs into the system. For a full list of this user's privileges, see figure 3.1.
- The 'user2' database user, who has a very limited set of privileges and is used for all of the system functions that are accessed by common visitors. For a full list of this user's privileges, see figure3.2.

Database Table	User Privileges
album	SELECT, INSERT, UPDATE, DELETE
comment	SELECT, INSERT, UPDATE, DELETE
fileformat	SELECT
image	SELECT, INSERT, UPDATE, DELETE
posts	SELECT, INSERT, UPDATE, DELETE
tag	SELECT, INSERT, UPDATE, DELETE
template	SELECT
user	SELECT, INSERT, UPDATE
usersactionlog	INSERT
website	SELECT, INSERT, UPDATE, DELETE

Figure 3.2: Privileges of user 'user1' for all the database tables.

Database	User Privileges
album	SELECT, INSERT, UPDATE
comment	SELECT, INSERT, UPDATE
fileformat	SELECT
image	SELECT, INSERT, UPDATE
posts	SELECT, INSERT, UPDATE
tag	SELECT
template	SELECT
user	SELECT, INSERT
usersactionlog	INSERT
website	SELECT, INSERT, UPDATE

Figure 3.3: Privileges of user 'user2' for all the database tables.

4. Interface Design

4.1 Introduction

Up to this point the reader has had the chance to get more familiar with the functionality that Retina-Destruction offers to its user base, as well as the structure of the database.

This chapter will analyze the my decisions design behind the design of the system's graphical interfaces. I tend not to use any ready templates, but to create designs that will match the functions of each system instead.

The designs of the system were implemented with the use of HTML elements, and CSS rules. No HTML tables are used anywhere in the layout. It is recommended that the reader views the video files that are included in the accompanying DVD.

After I determine the operations that I want each system to feature, I usually take a fresh look at an article by Jakob Nielsen (2005), titled “Ten Usability Heuristics”⁷.

These are ten general principles for user interface design. They are called "heuristics" because they are more in the nature of rules of thumb than specific usability guidelines (Nielsen, 2005).

I believe it is important to briefly list these Heuristics in this chapter, before moving on.

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help users recognize, diagnose, and recover from errors
10. Help and documentation

Figure 4.1: Jakob Nielsen's, “Ten Usability Heuristics”(2005)

4.2 The General Design

I settled on a very minimalistic text-based design that will not feature any images at all, and possibly be supported by many different web browsers. I knew that the system will have a lot of complicated functions that would establish many requests for data from both the server and the database, which would then go through very complex algorithms, possibly resulting in heavy loading times. Moreover, I have a great fascination with simple, clean designs that are common in print magazines and wanted to have a clean white background.

Since Retina-Destruction is in its core a tool that generates websites, I felt that the user should have the highest possible width and height of the browser window to work, and not feel crowded by unnecessary information. This excluded the option of having fixed width on the numerous pages of the system, but it also excluded the possibility of positioning the navigation links on the left or right of the layout.

Additionally, I needed the logo of the website, as well as the account options (sign-in, register) to be in a clear position, and not to fill too much space in terms of height. The most common place for that information is to be placed on the header and this is where I chose to position them too. I placed the navigation links underneath the logo, capitalized their text and increased their font size to make them more distinguishable.

My concern was how to fill the rest of the empty page where each section would load. The contents of the sections fills the entire page, but only once the user selects a navigation link. What fills the page before that when it first loads, or when no section is selected? Furthermore, because of the title “Retina-Destruction” I wanted to play around with the idea of visual noise, a state of unorganized text filling the page, but in a way that is not tiring to the users’ eye. In other words, I needed an organized and interesting way to include a very chaotic state of elements.

At this point I should refer to a website layout that helped me a lot with my decision: the promotional website of Norwegian musician and artist Magne Furuholmen. It was launched on September 2004⁴. This website had the lyrics of several songs filling the page in its entirety, with hidden links placed in various places.

The main body of the Retina-Destruction welcome page is filled with text. I incorporated the navigation links next to some text that introduces each section. When one of the links is selected, the page automatically scrolls to the corresponding section, morphing the text into a more presentable and comprehensive state, while loading the sub-page of the section. The rest of the sections fade-out and scroll out of the page. I achieved this with the use of a combination of AJAX libraries and CSS formatting.

⁴ This website is available at <http://www.magne-f.net/1/> and was developed by an agency called Sounds Like You (<http://www.soundslikeyou.no/>).

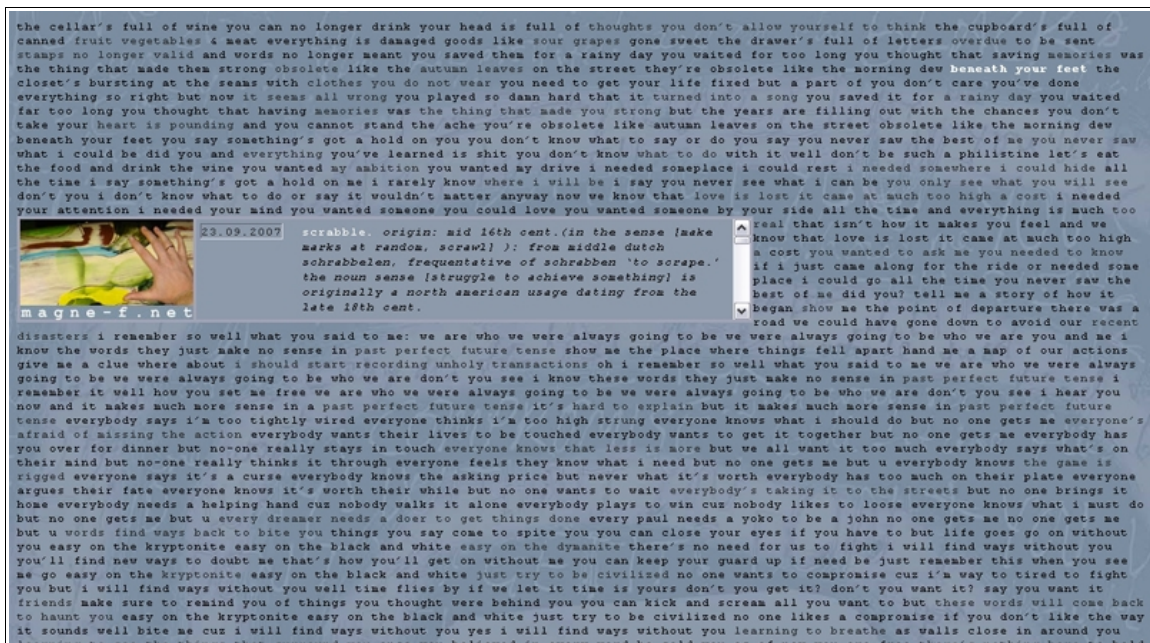


Figure 4.2: <http://www.magne-f.net/1/>

Some sections of the system have very limited informative text. In these cases I like to play around with the negative (empty) available space, and make the eye of the reader focus on the right section.

The main interface of the system contains introductory information about the system, links to generated websites and the “registration” and “sign-in” forms. The account page of the registered user features the same logic in the design, only now the navigation links are different and all the text that fills the body of the page is in larger font. I also numbered each section to help the user navigate to the sections that include the more necessary information.

The main interface can also be viewed in the video file “video01_main_website.avi”.

4.3 Designing the Account Sections

The most challenging sections where the 'album', 'blog' and 'design editor', since the registered artists will most commonly use these ones. Each section has a complicated set of parameters and options. By adopting a bad design I would end up making the whole process of creating a website feel unbearable to the users.

4.4 The Albums Section & Blog Section

These two sections have the same organization of their contents, since the information they require is similar.

The first page of the album section includes a list of all the image albums that the artist has created and one text button to add a new one. The artist can change the visibility status of each album (invisible albums are not viewed by the visitors), and view the comments that were left by visitors. The comments are loaded on a new layer, on top of the previous one, thus avoiding any unnecessary page loads.

When an album is selected, the edit page is loaded. This page features two subsections: one that contains all the forms and options to edit the album, and next to it a preview of the results of these changes in the way they will be viewed by the visitors of the website.

I added three different options on how the images of each album are presented, to give the registered artists more freedom in terms of presentation. The available options are:

- **Fullsize** where images are presented in their full dimensions.
- **Thumbnails** and **Large Thumbnails**, where the images are loaded in a new layer positioned at the top of the page. The opacity of the page dims and the images is presented in a fuller size, along with all the relevant information, without taking the user away from the album edit page.



Figure 4.3: A selected image presented on a new layer.

4.5 The Generated Website

In spite of my initial thoughts about the design of the generated websites, I created a layout template that is very generic. My aim was to dress the websites with a simple but interesting look, so that I can appeal to a larger demographic of users. Nonetheless, I selected a fair number of elements that could be edited from the “design editor” section of each account.

Each website is structured to consist of three segments. The first one is a very generic header which appears on all the websites and contains the logo of Retina-Destruction (which is also a link to the main page), including the 'register' and 'sign-in' links (when the user is not signed in), or a set of links that enable registered signed-in users to navigate to their account page, sign out or go to their website.

The second segment of the layout is the sidebar. This sidebar can contain the following:

- the logo of each website
- the date it was last edited
- the e-mail address of the artist
- a tag cloud with all the tags which characterize the images that belong to public albums (this tag cloud consist only of tags that the specific artist has inserted)
- the main navigation system with links to the **news** page, the pages for individual **albums**, the **about** page (which contains the profile information of the user), and the **contact** page. The links to albums that belong to the same category are grouped together. The contact page is used to send messages to the artist via e-mail.

The sidebar can be set to appear on the left, on the right, or even the top of the layout. If the artist chooses a “top” alignment, then only one link appears for all the albums. This link loads a page with all the public albums of the artist.

The last segment of the layout is the main section, which is always positioned at the center of the website. This is where all the content appears.

When a visitor of the website clicks on the **news** link, the five most recent blog entries are loaded in the main section, with links to the previous five posts. The blog entries appear with their complete information. The rest of the links have a similar effect on the main section.

If a blog entry or an album presents its included images in their full size, only the first five images appear, with a link at the bottom (“load more images”), which loads the rest of the images.

4.6 The Design Editor

The development of the design editor proved to be the most difficult section in the entire system. It contains 20 different parameters that are used by each registered artist to edit the design of their website. To help the artists understand all these parameters, I added four templates on the top of this page that essentially contain different combinations of these parameters.

The values of these parameters are stored in the database table titled 'website'. More information about the structure of this table can be found in Chapter 3.2 and Appendix D.

Figure 4.4: The design editor.

4.7 Different Browser Renderings

The layout designs, and AJAX functions of Retina-Destruction were tested in the latest versions of the web browsers Mozilla Firefox, Google Chrome, Opera and Safari. From the numerous tests of all its operations the system appears to work perfectly. I should note that during development I accessed the system through Google Chrome and Mozilla Firefox.

Internet Explorer appears to have many problems with the CSS instructions of the design that need to get fixed in the later iterations of the system. For example content that is loaded on new layers, such as the comments and the images, do not appear in their instructed positions. At this moment I can only assume that

there are problems with the “**z-index**” instructions for positioning this content on top of all the other website elements.

I also performed some tests in the mobile version of Safari. This browser has a very unique way of rendering websites.

“Mobile Safari uses a viewport to show you websites. Imagine a book in front of you. Take a piece of paper, cut a 320x416 square in it, and lay it over the book. To read the book, move the paper around and position the hole over the words you want to see. This is exactly what Mobile Safari’s viewport is doing. When you flick and scroll, you’re moving the viewport around while the website behind it stays static.

This renders fixed positioning null and void on iPhone. An element that has its position fixed is affixed to the body, not the viewport. So it is actually working as intended, though most people would prefer it attached to the viewport.”

(Richard Herrera, August 5, 2008, Fixed positioning in Mobile Safari, <http://doctyper.com/archives/200808/fixed-positioning-on-mobile-safari/>)

Moreover I noticed that the browser has problems when it renders elements of a page that use the “overflow” CSS instruction. This parameter “tells” the browser to either present, or hide a scroll bar for the contents of an HTML element of the page, when a certain defined height is exceeded.

For instance we can assume that there is an element containing text that needs 600 pixels worth of height to present itself normally. We use CSS to tell this element to have a height of 200 pixels and add the “overflow: scroll” instruction. This way the element will be presented in the defined height and the rest of the content will be visible when we use the scroll bar.

From my experience with this browser, the scroll bar for these elements will not appear and the users are unable to view all the content of the page. I needed to include several similar instructions for many elements in the Main and Account Interfaces of the systems, so the contents of these pages do not appear as intended.

However, the generated websites appear to work perfectly with only minor fixes to the positioning and the sizes of some elements. I included these instructions in a separate CSS file that loads only in the mobile versions of web browsers.

5. Code Authoring Methodologies & Files Organization

5.1 Introduction

This chapter is not meant to describe all my decisions and methods that on developing the entirety of the system. An attempt like this would take a lot more time than the actual production of the system and would surely tire the reader, as it requires very detailed descriptions of algorithms and code functions.

The structure of the Retina-Destruction system was designed in such a way as to limit any unnecessary re-loading of data that is exported from the database. I tried to write libraries of code functions which are used by the entirety of the system, and contain algorithms that are unique in the system.

Moreover, I wanted the generated websites to load only a subset of all the PHP files, to free the system from any redundant loading.

The PHP code

I separated the PHP files of the system in two types.

- The “controlling” PHP files which include all the functions that the system needs to communicate with the database and process all the data.
- The “presentation” PHP files that rely on the first ones to display the results.

The main “presentation” PHP files are stored in the root folder of the website and the rest in the **./rdpagecontents/** folder. The files included in the latter are appended via AJAX to the main files to produce dynamic information.

The (controlling) PHP files are organized primarily by the database table that their functions can access. For example all the PHP functions that are used to export data are included in the file **./rdincfiles/album.inc.php**. The ones that insert, update or delete data from the “**album**” table can be found in the **./rdincfiles/albumedit.inc.php** file. The same rule applies to the following files as well:

- **./rdincfiles/blog.inc.php**
- **./rdincfiles/blogedit.inc.php**
- **./rdincfiles/comment.inc.php**
- **./rdincfiles/commentedit.inc.php**
- **./rdincfiles/image.inc.php**
- **./rdincfiles/imageedit.inc.php**
- **./rdincfiles/user.inc.php**

- **./rdincfiles/useredit.inc.php**
- **./rdincfiles/website.inc.php**
- **./rdincfiles/websiteedit.inc.php**

It should be mentioned that only the files that select data from the database are required for the generated websites. However, these files also include the methods that update the views counter of the selected entries, since these are needed in the generated websites.

All the files that select entries from the database contain a central function with a very similar structure. For example the **./rdincfiles/album.inc.php** file contains a function called **album_get()** which has two parameters:

- **tokenKey**. If this parameter has the value 'aid', then the other parameter contains the id of the album in the database and the function returns an array with all the data of this album. If the parameters has the value 'uid' then the tokenValue parameter contains the id of a user. In this case the function returns an array with all the album data of all the albums belonging to this user.
- **tokenValue** The search value.

Other functions that have the same structure are:

- `image_get($tokenKey, $tokenValue)`
- `blog_get($tokenKey, $tokenValue)`
- `website_get($tokenKey, $tokenValue)`
- `comment_get($tokenKey, $tokenValue)`

From the rest of the “controlling” PHP files, the following are the most important:

- **./rdincfiles/functionsmapping.inc.php** This file is referenced by all the AJAX requests and depending on the request loads a different set of PHP files and calls the appropriate function.
- **./rdincfiles/rdconfig.inc.php** Defines all the variables that have a global scope in the system
- **./rdincfiles/responses.inc.php** When a PHP function is called by an AJAX request and needs to send back the results, a JSON object is created. This file contains functions that produce JSON objects for all the operations of the system.
- **./rdincfiles/tbdbase.class.inc.php** A PHP class file which according to the system operation uses a different database user to establish the connection with the database. It also includes functions that execute SELECT, INSERT, UPDATE and DELETE queries, which return to the calling function, the results in form of an array.
- **./rdincfiles/validate.class.inc.php** A PHP class file which contains methods that check and validate submitted data from all the operations of the system

The registered users folder

For security reasons whenever a new user fills the registration form, the system checks:

- if the entered e-mail account exists in the database
- if the database already contains a user with the same username
- if the username is the same with any file and folder name of the system
- if the username is a default filename like 'index.php', 'index.html', and so on.

If after all these checks the provided username proves to be unique in the system, a new folder with the username is created inside the **./rdusers/** and **./root/** folders. Additionally an index.php file is created inside this folder that redirects the browser to the main index.php file in the root folder of the system, passing the username as a parameter.

For example, if a user enters the following address in the browser

<http://dimitrioschantzis/rdsystem/rdusers/jamesdoe/>

then the browser is redirected to the page

<http://dimitrioschantzis/rdsystem/index.php?jamesdoe>

which in turn will redirect to the page that contains the portfolio of that user

<http://dimitrioschantzis/rdsystem/portfolioindex.php?jamesdoe>

The registered users folder contain all the images that they upload to their website. For instance this is the folder structure inside the folder **./rdusers/jamesdoe** witch belongs to the user jamesdoe:

- **./rdusers/jamesdoe/coverimages/**
- **./rdusers/jamedoe/designimages/**
- **./rdusers/jamesdoe/portfolioimages/**
- **./rdusers/jamesdoe/portfolioimages/largethumbnails/**
- **./rdusers/jamesdoe/portfolioimages/thumbnails/**

The images contained in the last three folders all share the same file name since they are different versions of the same image. This is very helpful approach because I do not have to save the file names for each different version in the database.

A more important reason is evident in the case of an image thumbnail getting selected anywhere in the system. As I mentioned in a previous chapter this event causes a layer to appear on top of the website which contains the image in full size. This previously not-loaded image is presented to the user without any AJAX requests to the server to fetch more data.

This is achieved using very basic AJAX methods. The source of every image that is loaded in an HTML

page, points to the file path of the image in the server. When a thumbnail is selected I can take the path of it and from that compile the path of the full size version. For example, the path of a thumbnail image from which I want to generate the path of it full sized version is:

./rdusers/jamesdoe/portfolioimages/thumbnails/20100806194738.jpg

using javascript code, I break this string in two parts. The first one is the string before the word 'thumbnails':

./rdusers/jamesdoe/portfolioimages/

and second one is:

/20100806194738.jpg

Since all the different version of an image share the same file name, I just add the word 'fullresolution' between them, and the resulting string is:

./rdusers/jamesdoe/portfolioimages/fullsize/20100806194738.jpg

which gives the full size version of the image.

6. System Security

6.1 Introduction

PHP is an exceptional programming language for developing interactive web applications fast. It is based on the C programming language, thus developers with experience in C, C++ or JAVA, can very easily familiarize themselves with its methods and logic. It also includes a large number of features that friendly to even the most unfamiliar programmers. However, some of these features can lead to great security problems for the developed application.

PHP can be equally secure with any other language, provided that the basic types of errors are understood by the developers.

The security of a system has to be a clear and coherent process and the acceptance of a safe approach during the implementation of a web application can allow the developers to produce more coherent and strong code functions.

This chapter presents the ten most common and serious security risks of web applications according to the OWASP⁹ (Open Web Application Security Project) community and how Retina-Destruction is secured against them by following the guidelines of OWASP and some personally authored algorithms.

6.2 The OWASP Top 10 2010

This chapter includes the ten most serious security risks of web applications as described by the OWASP. I decided to include a list of these risks as referenced in the OWASP website⁹ and documents¹⁰.

1. Injection
2. Cross-Site Scripting (XSS)
3. Broken Authentication and Session Management
4. Insecure Direct Object References
5. Cross-Site Request Forgery (CSRF)
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Redirects and Forwards

6.3. Security in Retina-Destruction

Session Management

The operations of Retina-Destruction rely heavily on session variables instead of cookies, because the latter are stored in the computer of the user.

In the beginning of every page the system checks if the user has the necessary credentials to browse the contents of the page. For example unregistered users do not have any access the the “Account” section and the subsequent sub-sections.

When a user attempts to sign-in, the system checks:

- If the user is the administrator of the system.
- If the user is registered to the system and if his/her account is activated.

The IP address is not taken into account for the sign-in of a user to the system.

After the successful sign-in of the administrator the system fills the appropriate \$_SESSION variables, but also creates a 2D array filled with 63 dummy combinations of username and passwords. The credentials of the administrator are hidden inside this array in a random position. For every action that requires the administrator's credentials, the system calculates the position of the credentials in the array and checks them. If these are correct then the action is executed, if not the connection is terminated and the user gets signed-out.

When I started the development of the system I included to the beginning of each php page the following lines of code that reset the session ID,

```
session_regenerate_id(true);
if(!version_compare(PHP_VERSION(), "5.1.0", ">=")) {setcookie( session_name(), session_id(),
ini_get("session.cookie_lifetime"), "/" );}
```

but because any kind of reset to the session ID presented many problems with the library that uploads images to the file server, I had to remove them.

The system does not support a “Remember Me” function that would enable a registered user to automatically sign-in each time he visits the system. All the session variables are destroyed when the user signs-out of the system.

It should be mentioned that the users passwords are first encrypted and then saved in the database. The user might have defined a secret keyword for password, but this keyword is used to produce a code that is the one stored. When the user tries to sign-in, the keyword is encrypted again to produce the code and compared with the one in the database.

Protection against XSS and Injection flaws

To protect the system and the integrity of the database against XSS and injection errors, all data that is submitted from the user, are inserted to three different functions:

- **addslashes()**. This function returns a string that places the “\” (backslash) character before any instance of the characters “'” (single quote), “”” (double quotes), “\” (backslash) and NUL (the NULL byte). The returned string can be safely used in a query.
- **htmlentities()**. Some characters have a special meaning in HTML, and must be represented with HTML-type entities so that can maintain their meaning.

For example, the “<” (less than) and “>” (greater than) characters, are used to define an HTML element, such as <div> or . If a user submits in a text field the “<” character, and this text field later gets embedded in the HTML structure of a page, then it might cause some trouble in the rendering of that page.

This function returns these characters in a form that does not threat the structure of the page. Some of these transformations, are:

- “&” becomes “&,”
- “”” becomes “",”
- “”” becomes “',”
- “<” becomes “<,”
- “>” becomes “>,”

It is very important for a function to escape these characters. The next example explains the reason. Consider a <textarea> field of a form, where a user can type any kind of text that is most commonly stored in a database. If the user types a string like:

```
<script type='javascript'>
    window.location.replace("http://dimitrioschantzis.com");
</script>
```

then it will be saved in the database without escaping the “dangerous” characters. When this string is exported from the database and appended to the HTML structure of the page, it will immediately redirect the browser to the website “<http://dimitrioschantzis.com>”.

This might not seem as a very serious threat but if the attacker appends to this address a call to a javascript function that gets all the session (or cookie) variables that are set in the website, then when the browser is redirected to the “malicious” website it will also send all the session variables and expose some crucial information.

- **Trim()**. This function deletes any number of empty spaces before and after the supplied string.

Additionally the system uses function that check if all the necessary form fields are submitted and each field

has the appropriate kind of value (number, characters, regular expressions). Also it checks if every field value includes the appropriate number of characters.

Protection Against Malicious File Executions

Every function in the system is organized in specific files. All these files are called inside one function, using the `require()` function. This “central” file acts like a “function map” and according to the request that it needs to forward, it “requires” only the appropriate files for that action.

The `require()` function can be very harmful to a system, if a user input is directly used for the selection of the needed file. For example the following line of code is **extremely** dangerous:

```
require("./rdincfiles/anotherdirectory/" . $_GET['name_of_file']);
```

If a malicious user enters a string such as “../././index.php” then the `require` function, will attempt to find the `index.php` file that is placed in the directory that includes the “rdincfiles” folder. Such a file might not exist, but if the user continues to try different paths and different file names, then he might learn information about the file organization of the system.

User input is not to be directly used to call the appropriate files. In Retina-Destruction, every request is sent in the form of a code that is matched to the appropriate available action:

```
reset($_GET);
if(isset($_GET['type']))
{
    if(!preg_match("/^[0-9]([0-9]*)/", $_GET['type'])) { $_GET['type'] = NULL; }
    else { $get_type = $_GET['type']; }
    unset($_GET['type']);
} else { $get_type = NULL; }

switch($get_type)
{
    case 3:
        require('useredit.inc.php');
        user_edit('insert','ajax');
        break;
    case 4:
        require('useredit.inc.php');
        user_edit('update','ajax');
        break;
```



```

...
...
} //end switch

```

In the above example, the system first checks if a `$_GET` variable with the name 'type' exists and then if its value is a number. If none of these cases is true then the request is rejected. In any other case, the value is used to load the appropriate file and then call the wanted function.

Protection against CSRF flaws

Every HTML `<form>` element most usually contains the following two parameters:

method: the method which is used to send the data to the server

action: the file that is used to handle this data.

If the form data is sent using the **get** method, then the data are sent to the server in a very transparent way (unless SSL is used). Moreover if a malicious user reads the source rendered code of a page (which is by default accessible by all web browsers) then he/she can easily locate the path and name of the file that handles the specific form. From there he/she can start sending any kind of data to that file, and watch the results, or even threat the integrity of the database.

In Retina-Destruction the `<form>` elements do not contain these parameters, and in some cases do not exist at all. All the forms are submitted via AJAX, and the requests are sent to a central file (`./rdincfiles/functionsmappinc.inc.php`) using the **post** method. From then on, the appropriate function is called to handle the data.

Each function checks if the data are sent using the **post** method, if not then the request is terminated. Then it checks if the form was submitted using the expected way, which is the case where the user submits the form by filling the fields manually and then pressing the “submit” button.

These checks can be easily bypassed, so each form when it gets submitted generates a specific code for the function that needs to be executed. This code is recreated in the file that controls the request and then these two are compared. If they do not match, then the request is aborted.

For added protection each form sends a second parameter containing the name of the page that includes the form.

Information Leakage and Improper Error Handling.

It is very important for a system to handle every error that might appear in its operation in a manner that informs the user about the error, but without giving too much information about its functions and structure.

In Retina-Destruction all the usual cases than an error might happen, are handled in a timely manner to protect the database data, and the system's integrity. If a serious attack is detected then the user is immediately logged out. It would be a very good idea to implement a function that also catalogs the IP address of the attacker and using that parameters, prevents any access to the system for a predetermined period of time.

There are a lot more smaller ways that the system is protected against errors and attacks. My main concern was to mention the most important ones, that are shared throughout all the functions of the system.

The only main concern with the submitted version of the system is that due to time limitations, attack attempts and system errors are not stored in the database or in a log file, even though I designed the structure of all the appropriate methods. Right now the system prevents any errors and attacks to harm the database and the structure of the system, but also informs the users that an error has happened without giving away too much information.

Appendices

Appendix A: DVD CONTENTS

./READ_ME.rtf – File:

Explanation of the DVD Contents. Includes the Appendix A of this publication in its entirety

./SYSTEM_FILES/ - Folder

Contains all the necessary files and folders for the system installation.

./SYSTEM_FILES/DATABASE/rddb.sql - File

Contains SQL code for creating the database of the system, and the database users with all the appropriate privileges. This code file includes test data for all the database tables. The test data are not included in this publication.

NOTE: Execute the query that creates the database on its own before the rest of the queries. After you execute this query select the newly created database and then run the rest of the queries.

./SYSTEM_FILES/SOURCE/rdsystem/ - Folder

Includes the code files of the system. To guarantee the normal operation of the system, this folder structure as well as the file names, should not be altered in any way.

./DISSERTATION/[s0972736]DISSERTATION[DIMITRIOS_CHANTZIS].pdf – File

File that contains the written part of the dissertation.

./IMAGE_RESOURCES/ - Folder

Contain test images for the system generated portfolio website “James Doe”. These images are used for demonstration purposes in various screenshots of the system, and in the documentation videos. All images were illustrated by Dimitrios Chantzis.

./SYSTEM_SCREENSHOTS/ - Folder

All the screenshot images of the system that are featured in this publication.

./VIDEO_DEMOS/ - Folder

Contains all the demonstration videos of Retina-Destruction.

Appendix B: Installation Guide

B.1 Introduction

RetinaDestruction was authored using XHMTL, PHP, JavaScript (AJAX library JQuery, version 1.4.2), CSS and the relational Database MySQL. The system was developed in PHP 5, Apache 2 and MySQL 5. Most recent system operation checks were executed in PHP 5.2.5, Apache 2.2.6 and MySQL 5.0.45.

For the installation of these technologies to my local system, I used WampServer for MS Windows (versions XP and Vista), and XAMPP for Linux (Ubuntu 9.04).

More information about the reasons behind my decision to use these technologies, can be found in the “Introduction” chapter of this publication.

This guide refers to installation of the appropriate software to run the RetinaDestruction system in a local server (localhost).

B.2 WampServer & XAMPP

WampServer is an environment for developing web-based applications, using Apache, PHP and MySQL, including software for managing relational databases (PHPMyAdmin and SQLiteManager).

The most recent version is available at <http://www.wampserver.com/en/download.php> .

A detailed installation guide for WampServer is provided at <http://www.wampserver.com/en/presentation.php> .

Similarly to WampServer, XAMPP helps with the intallation of Apache, MySQL and PHP, as well as Perl. Currently there are XAMPP distributions for Linux, Windows, Mac OS X and Solaris. The most recent versions of XAMPP are available at <http://www.apachefriends.org/en/xampp.html> . For each distribution, there is a detailed installation guide.

B.3 Settings for the php.ini file

The php.ini file contains all the setting for PHP, and there are numerous copies of it after the installation of WampServer or XAMPP.

Depending on the software that was used to install Apache, MySQL and PHP, the php.ini file that needs to be updated is sited in a different location:

In the case of WampServer for MS Windows, the file is in under the path, /wamp/bin/apache/apache2.x/ (where x is the version of Apache2).

For XAMPP (MS Windows distribution), the file is in the folder /xampp/apache/bin/ .

For XAMPP (Linux distribution), the file is in the folder /opt/lampp/etc/ .

The following changes must be made in the php.ini file:

```
file_uploads = On
upload_max_filesize = 50M
max_input_time = 60
memory_limit = 128M
max_execution_time = 30
post_max_size = 8M
short_open_tag = On
```

Retina-Destructions sends e-mail messages for certain system actions (new account activation, reset account password request, visitor comments, contact users and administrator forms). In order to run these functions the following values must be reset-ed in the php.in file:

```
SMTP = mail.yourdomainname.com //add your domain name
smtp_port = 2626 // this value can differ, consult your web hosting provider
sendmail_from = mail@yourdomainname.com //add your domain name.
```

Changes to values in the php.ini file that regard the system's security are not required for the safe operation of RetinaDestruction. These values are mentioned in the chapter titled “System Security “ of this publication. A restart of the WampServer (or XAMPP) services is required to activate the above changes.

IMPORTANT: The Retina-Destruction application has been checked in PHP version 5.2.5. If the system is installed in a server with an earlier version of PHP, there is a chance that the above values will be different. In this case the application may not function properly.

B.4 Creating the Database and the Database Users

The Retina-Destruction application uses a database with the name “rddb”, which has to be created first. Using phpmyadmin (or any other software for managing MySQL), a large number of queries must be executed:

- The SQL code for constructing the database and the database table.
- The code for creating the 3 database users the system needs to access the database, and applying the proper privileges for each database table.
- Test data for the database table.

These queries are included in the `./SYSTEM_FILES/DATABASE/rddb.sql` of the DVD, and in Appendix C of this publication

The resulting database has the name “rddb” and the users:

- 'rdadmin' with password 'e97a2d7a33cbd2',
- 'user1', with password 'candy01'
- 'user2', with password 'candy02'

NOTE: The above usernames and passwords are provided ONLY for testing purposes and should are not used in the live version of the system. User credentials like the above are very easy to be guessed with dangerous results to the system's and the database's integrity.

The above user credentials should match the ones defined in the system file

`./SYSTEM_FILES/SOURCE/rdsystem/rdincfiles/rdconfig.inc.php` :

- Line 40: The `TBDB_DATABASE` value contains the database name.
- Line 41: The `TBDB_REGISTERED_USER` value contains the username of the first database user, “user1”.
- Line 42: The `TBDB_REGISTERED_PASSWORD` value contains the password for the first database user, “user1”.
- Line 43: The `TBDB_COMMON_USER` value contains the username of the second database user, “user2”.
- Line 44: The `TBDB_COMMON_PASSWORD` value contains the password for the second database user, “user2”.

NOTE: The administrator database account, “admin”, does not and should not appear in any system file, for security purposes.

B.5 Installing Retina-Destruction

The installation process of the system is very brief and easy. You can just copy the contents of the `/SYSTEM_FILES/SOURCE/` folder of the DVD, and paste them in the appropriate folder, named “www”. In the case of WampServer (MS Windows environment), the folder path is usually `c:\wamp\www`. For XAMPP (Linux environment), the folder path is usually `/opt/lampp/htdocs/`.

Thereafter, the installation folder path should match the one defined in the system file

./SYSTEM_FILES/SOURCE/rdsystem/rdincfiles/rdconfig.inc.php :

- Line 46: TBPARENT_DIR value should be /wamp/www/rdsystem/
- Line 47: TBUSERS_DIR value should be /wamp/www/rdsystem/rdusers/
- Line 48: TBRAW_DIR value should be /wamp/www/rdsystem/rdbin/

If you have configured the php.ini file for testing the system in localhost, then you should the following line should be edited too:

- Line 128: Should have the value 'disabled'.

Note: This is a system based variable, and in the case of 'disabled' the system continues will notify to the users that the emails have been sent normally.

Do not forget to set a default email account for sending notifications!

- Line 71: Should have the default email.

Lastly:

- Line 72: The value should include the server name. If the hosting address is retinadestruction.com, then:

```
define('SERVER_NAME','retinadestruction.com');
```

If it is rdsystem, then:

```
define('SERVER_NAME','rdsystem');
```

NOTE: If the system is installed in a Linyx Operating System, the above folder structure in the rdcnfig.inc.php file may differ. Also, the appropriate acces privileges must be assigned to all the files and folders.

The system is now ready to be used. You can access it from the address “<http://localhost/rdsystem/>”

Appendix C: Database & Database Users SQL Code

C.1 Database SQL Code

C.1.1 Creating the Database:

```
CREATE DATABASE `rddb` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

C.1.2 Structure for table 'album'

```
CREATE TABLE `album` (
  `aid` int(11) NOT NULL auto_increment,
  `uid` int(11) default NULL,
  `name` varchar(105) NOT NULL,
  `description` text,
  `embeddedvideos` text,
  `tid` int(11) NOT NULL,
  `coverid` int(11) default NULL,
  `imagesorder` text,
  `visibilitystatus` enum('visible','invisible') NOT NULL,
  `imageview` enum('fullsize','thumbnails','largethumbnails') NOT NULL,
  `views` int(11) default '0',
  `commentscounter` int(11) default '0',
  `type` enum('regular','blogpostsimagesalbum','coverpageimagesalbum') default 'regular',
  `imagescounter` int(11) default '0',
  `creationtimestamp` datetime NOT NULL default '0000-00-00 00:00:00',
  `lastupdatedtimestamp` datetime NOT NULL default '0000-00-00 00:00:00',
  PRIMARY KEY (`aid`),
  KEY `uid` (`uid`),
  KEY `tid` (`tid`),
  KEY `coverid` (`coverid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;
```

C.1.3 Structure for table 'comment'

```
CREATE TABLE `comment` (
  `cid` int(11) NOT NULL auto_increment,
  `pid` int(11) default NULL,
  `uid` int(11) default NULL,
  `aid` int(11) default NULL,
```



```

`name` varchar(255) NOT NULL,
`email` varchar(105) NOT NULL,
`body` text NOT NULL,
`submissiontimestamp` datetime NOT NULL default '0000-00-00 00:00:00',
PRIMARY KEY (`cid`),
KEY `uid` (`uid`),
KEY `pid` (`pid`),
KEY `aid` (`aid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

```

C.1.4 Structure for table 'fileformat'

```

CREATE TABLE `fileformat` (
  `fid` int(11) NOT NULL auto_increment,
  `extension` varchar(5) NOT NULL,
  `description` text,
  `type` varchar(100) default NULL,
  `mimetype` varchar(80) NOT NULL,
  PRIMARY KEY (`fid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

```

C.1.5 Structure for table 'image'

```

CREATE TABLE `image` (
  `iid` int(11) NOT NULL auto_increment,
  `uid` int(11) NOT NULL,
  `aid` int(11) NOT NULL,
  `pid` int(11) default '0',
  `fid` int(11) NOT NULL,
  `tags` text collate utf8_unicode_ci,
  `views` int(11) default '0',
  `caption` text collate utf8_unicode_ci,
  `filename` varchar(105) collate utf8_unicode_ci NOT NULL,
  `filesize` int(11) default NULL,
  `fileurl` varchar(255) collate utf8_unicode_ci default NULL,
  `imagetype` enum('regular','usercover','albumcover') collate utf8_unicode_ci default NULL,
  `uploadtype` enum('fileserver','database') collate utf8_unicode_ci NOT NULL,

```

```

`orientation` enum('horizontal','vertical') collate utf8_unicode_ci default NULL,
`lastupdatedtimestamp` datetime NOT NULL default '0000-00-00 00:00:00',
`submissiontimestamp` datetime NOT NULL default '0000-00-00 00:00:00',
PRIMARY KEY (`iid`),
KEY `uid` (`uid`),
KEY `aid` (`aid`),
KEY `fid` (`fid`),
KEY `pid` (`pid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT=1;

```

C.1.6 Structure for table 'posts'

```

CREATE TABLE `posts` (
  `pid` int(11) NOT NULL auto_increment,
  `uid` int(11) NOT NULL,
  `headline` varchar(105) NOT NULL,
  `body` text NOT NULL,
  `embeddedvideos` text,
  `type` enum('imagesupdate','newspost') NOT NULL,
  `tags` text,
  `images` text,
  `views` int(11) default '0',
  `commentscounter` int(11) default '0',
  `imageview` enum('fullsize','thumbnails','largethumbnails') NOT NULL,
  `visibilitystatus` enum('visible','invisible') default 'visible',
  `lastupdatedtimestamp` datetime NOT NULL default '0000-00-00 00:00:00',
  `creationtimestamp` datetime NOT NULL default '0000-00-00 00:00:00',
  PRIMARY KEY (`pid`),
  KEY `uid` (`uid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

```

C.1.7 Structure for table 'tag'

```

CREATE TABLE `tag` (
  `tid` int(11) NOT NULL auto_increment,
  `uid` int(11) NOT NULL,

```

```

`name` varchar(105) NOT NULL,
`description` text,
`type` enum('image','profile','post','album') NOT NULL,
PRIMARY KEY (`tid`),
KEY `uid` (`uid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

```

C.1.1.8 Structure for table 'template'

```

CREATE TABLE `template` (
  `tid` int(11) NOT NULL auto_increment,
  `name` varchar(80) NOT NULL,
  `sidebarinfo` varchar(255) character set utf8 collate utf8_unicode_ci default NULL,
  `sidebarorientation` varchar(255) character set utf8 collate utf8_unicode_ci default NULL,
  `imagetagcloud` enum('yes','no') character set utf8 collate utf8_unicode_ci default 'yes',
  `designcsscode` longtext,
  `designformoptions` longtext,
  PRIMARY KEY (`tid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

```

C.1.1.9 Table structure for 'user'

```

CREATE TABLE `user` (
  `uid` int(11) NOT NULL auto_increment,
  `name` varchar(255) NOT NULL,
  `username` varchar(255) NOT NULL,
  `password` varchar(105) NOT NULL,
  `email` varchar(105) NOT NULL,
  `avatar` int(11) default NULL,
  `views` int(11) default '0',
  `tags` text,
  `gender` enum('male','female') default 'male',
  `description` text,
  `facebook` varchar(105) default NULL,
  `myspace` varchar(105) default NULL,
  `twitter` varchar(105) default NULL,
  `youtube` varchar(105) default NULL,

```

```

`vimeo` varchar(105) default NULL,
`newsletter` enum('enabled','disabled') default 'disabled',
`featured` enum('true','false') default 'false',
`albumcomments` enum('enabled','disabled') default 'enabled',
`blogpostcomments` enum('enabled','disabled') default 'enabled',
`commentnotifications` enum('enabled','disabled') default 'enabled',
`visibilitystatus` enum('visible','invisible') default 'visible',
`accountstatus` enum('active','nonactive','hardnonactive') NOT NULL,
`registrationstatus` enum('complete','incomplete') default NULL,
`lastupdatedtimestamp` datetime NOT NULL default '0000-00-00 00:00:00',
`registrationtimestamp` datetime NOT NULL default '0000-00-00 00:00:00',
PRIMARY KEY (`uid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

```

C.1.10 Table structure for table 'usersactionlog'

```

CREATE TABLE `usersactionlog` (
  `alid` int(11) NOT NULL auto_increment,
  `uid` int(11) default NULL,
  `message` text collate utf8_unicode_ci NOT NULL,
  `entrytype` varchar(10) collate utf8_unicode_ci NOT NULL,
  `actiondatetime` datetime NOT NULL default '0000-00-00 00:00:00',
  `userid` varchar(25) collate utf8_unicode_ci NOT NULL,
  `userprivileges` varchar(50) collate utf8_unicode_ci default NULL,
  PRIMARY KEY (`alid`),
  UNIQUE KEY `actiondatetime` (`actiondatetime`),
  KEY `user_id` (`uid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT=1 ;

```

C.1.11 Table structure for table 'website'

```

CREATE TABLE `website` (
  `wid` int(11) NOT NULL auto_increment,
  `uid` int(11) NOT NULL,
  `title` varchar(105) collate utf8_unicode_ci NOT NULL,
  `urltitle` varchar(105) collate utf8_unicode_ci NOT NULL,
  `views` int(11) default '0',

```

```

`tid` int(11) default '0',
`wallpaper` varchar(255) collate utf8_unicode_ci default NULL,
`logo` varchar(255) collate utf8_unicode_ci default NULL,
`sidebarinfo` varchar(255) collate utf8_unicode_ci default NULL,
`sidebarorientation` varchar(255) collate utf8_unicode_ci default NULL,
`imagetagcloud` enum('yes','no') collate utf8_unicode_ci default 'yes',
`designncsscode` longtext collate utf8_unicode_ci,
`designformoptions` longtext collate utf8_unicode_ci,
`uploadimageswidth` varchar(5) collate utf8_unicode_ci default '800',
`uploadimagetype` enum('1','2','3') collate utf8_unicode_ci default '3',
`coverpage` enum('empty','blogsection','randomimage') collate utf8_unicode_ci default 'empty',
`visibilitystatus` enum('visible','invisible') collate utf8_unicode_ci default 'visible',
`status` enum('active','nonactive') collate utf8_unicode_ci default 'active',
`lastupdatedtimestamp` datetime NOT NULL default '0000-00-00 00:00:00',
`creationtimestamp` datetime NOT NULL default '0000-00-00 00:00:00',
PRIMARY KEY (`wid`),
KEY `uid` (`uid`),
KEY `tid` (`tid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci AUTO_INCREMENT=1 ;

```

C.2 Creating the Database users

C.2.1 user1

Code for creating the Database User that will be used for all the database transactions of a registered users. These transactions happen ONLY through using the RetinaDestruction application, and never straight from a relational database managing system.

```

CREATE USER 'user1'@'localhost' IDENTIFIED BY 'candy01';
GRANT USAGE ON * . * TO 'user1'@'localhost' IDENTIFIED BY 'candy01' WITH
MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0
MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0 ;
GRANT SELECT, INSERT, UPDATE, DELETE ON `rddb`.`album` TO 'user1'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON `rddb`.`comment` TO 'user1'@'localhost';
GRANT SELECT ON `rddb`.`fileformat` TO 'user1'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON `rddb`.`image` TO 'user1'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON `rddb`.`posts` TO 'user1'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON `rddb`.`tag` TO 'user1'@'localhost';

```

```
GRANT SELECT ON `rddb`.`template` TO 'user1'@'localhost';
GRANT SELECT, INSERT, UPDATE ON `rddb`.`user` TO 'user1'@'localhost';
GRANT INSERT ON `rddb`.`usersactionlog` TO 'user1'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON `rddb`.`website` TO 'user1'@'localhost';
```

C.2.2 user2

Code for creating the Database User that will be used for all the database transactions of all common users (visitors). These transactions happen ONLY through using the RetinaDestruction application, and never straight from a relational database managing system.

```
CREATE USER 'user2'@'localhost' IDENTIFIED BY 'candy02';
GRANT USAGE ON *.* TO 'user2'@'localhost' IDENTIFIED BY 'candy02' WITH
MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0
MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0 ;
GRANT SELECT, INSERT, UPDATE ON `rddb`.`album` TO 'user2'@'localhost';
GRANT SELECT, INSERT, UPDATE ON `rddb`.`comment` TO 'user2'@'localhost';
GRANT SELECT ON `rddb`.`fileformat` TO 'user2'@'localhost';
GRANT SELECT, INSERT, UPDATE ON `rddb`.`image` TO 'user2'@'localhost';
GRANT SELECT, INSERT, UPDATE ON `rddb`.`posts` TO 'user2'@'localhost';
GRANT SELECT ON `rddb`.`tag` TO 'user2'@'localhost';
GRANT SELECT ON `rddb`.`template` TO 'user2'@'localhost';
GRANT SELECT, INSERT, UPDATE ON `rddb`.`user` TO 'user2'@'localhost';
GRANT INSERT ON `rddb`.`usersactionlog` TO 'user2'@'localhost';
GRANT SELECT, INSERT, UPDATE ON `rddb`.`website` TO 'user2'@'localhost';
```

C.2.3 rdadmin

Code for creating the Database user that will be used for all the database transactions of the system administrator. These transactions happen ONLY through using the RetinaDestruction application, and never straight from a relational database managing system.

```
CREATE USER 'rdadmin'@'localhost' IDENTIFIED BY 'e97a2d7a33cbd2';
GRANT USAGE ON *.* TO 'rdadmin'@'localhost' IDENTIFIED BY 'e97a2d7a33cbd2' WITH
MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0
MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0 ;
GRANT ALL PRIVILEGES ON `rddb`.* TO 'rdadmin'@'localhost' WITH GRANT OPTION ;
```

C.2.4 Important INSERT queries

```
INSERT INTO `fileformat` (`fid`, `extension`, `description`, `type`, `mimetype`) VALUES
```


Appendix D. Database Tables & Demonstration Entries

D.1 Table "album"

Field Name	Description
aid (primary key)	The album entry unique id.
uid	The id of the user who owns this album.
name	The title of the album.
description	A brief description of the album and its contents.
embeddedvideos	Emved code for videos that are hosts in youtube.com or vimeo.com.
tid	The id of a tag stored in the 'tag' table. Each album can have only one tag, which is references in the system as "category".
coverid	The id of an image stored in the 'image' table.
imagesorder	A string that contains all the images this album includes the order specified by the user.
visibilitystatus	Each album can either be "visible" or "invisible" to the visitors of the website.
imageview	Value that stores the way in which images are presented in an album. The possible values are 'fullsize', 'thumbnails' and 'largethumbnails'.
view	A counter of how many times this album has been viewed.
commentscounter	A counter value for all the comments written for this album
type	The type of images that this album stores. Possible values are 'regular', 'blogpostimages' and 'coverpageimages'.
imagescounter	A counter of all the images stored in the album. In the current version of the system, this field is not used, since the same information can be obtained by counting the images in the 'imagesorder' field.
creationtimestamp	The date and time this album was created
lastupdatedtimestamp	The date and time this album was last updated

Figure D.1: Structure of database table 'album'.

Field Name	Value
aid	232
uid	5
name	Illustration

description	This album contains some of my illustration work
embeddedvideos	null
tid	100
coverid	484
imagesorder	428:.....:429:.....:430:.....:
visibilitystatus	visible
imageview	largethumbnails
views	74
commentscounter	5
type	regular
imagescounter	0
createtime	'2010-08-05 18:44:35'
lastupdatedtimestamp	'2010-08-13 21:22:09'

Figure D.2: Demonstration entry for database table 'album'.

D.2 Table 'comment'

Field Name	Description
cid (primary key)	The comment entry unique id
pid	The blog entry id that this comment refers to. If the comment refers to an album, this field has the zero (0) value.
uid	The user id who owns the album, or blog entry.
aid	The album id that this comment refers to. If the comment refers to a blog entry, then this field has the zero (0) value.
name	The name of the person writing the comment.
email	The email address of the person who left the comment
body	The message
submissiontimestamp	the date and time that the comment was saved

Figure D.3: Structure of database table 'comment'.

Field Name	Value
cid	129
pid	0
uid	5
aid	50
name	dimitrios chantzis
email	chantzis.dimitrios@gmail.com
body	Nice work on this album!!!
submissiontimestamp	'2010-08-13 21:23:58 '

Figure D.4: Demonstration entry for database table 'comment'.

D.3 Table 'fileformat'

This table is used to save all the acceptable file formats that the registered users can upload. In the current version of the system, this table only stores the accepted formats for the images that the users upload. If a user tries to upload an image with a file format not included in this table, the system rejects it.

Field Name	Description
fid (primary key)	The format entry unique id
extension	The extension of the format.
description	A brief description of the format
type	They type of the file. Possible values can be “image” or “document”. Currently the system only stores image file formats.
mimetype	The mimetype of the file

Figure D.5: Structure of database table 'fileformat'.

Field Name	Value
fid	1
extension	jpg
description	JPG image
type	image
mimetype	image/jpeg

Figure D.6: Demonstration entry for database table 'fileformat'.

D.4 Table 'image'

I should note that this table is designed to store information about images that are either saved in a file server, or saved in this table in binary form. Although the current system does not support functions to store images in the database, I decided to keep this structure just in case.

This is the reason that the fields titled 'filename' and 'fileurl' have the same value. In the case of images stored in the table, this 'filename' would be the name of the file presented to the user, and the 'fileurl' the name of the file as it is stored in the file server.

Field Name	Description
iid (primary key)	The entry unique id of the image
uid	The id of the user how has uploaded the image
aid	The id of the album where the image is included
pid	The blog entry id where the image also appears
fid	The file format of the upload image
tags	A string containing the ids of all the tags the user has specified for the image
views	A counter of how many times this image has been viewed.
caption	A brief description of the image
filename	The filename of the image
filesize	The size of the image
fileurl	The url of the image in the server. This is has the same value as the filename.
imagetype	The type of the image. Possible values are 'regular', 'usercover' and 'albumcover'.
uploadtype	Specifies how the images are saved in the system. Possible values are 'fileserver' and 'database'.
orientation	The orientation of the image file. Possible values 'horizontal' and 'vertical'.
lastupdatedtimestamp	The date and time this image entry was last edited
submissiontimestamp	The date and time this image entry was created

Figure D.7: Structure of database table 'image'.

Field Name	Value
iid (primary key)	414
uid	5
aid	232
pid	83
fid	1
tags	105:::103:::
views	11
caption	hero cat! fighting for truth, justice and dry food...
filename	20100805185901.jpg
filesize	705548
fileurl	20100805185901.jpg
imagetype	regular
uploadtype	fileserver
orientation	vertical
submissiontimestamp	'2010-08-05 18:59:07'
lastupdatedtimestamp	'2010-08-05 19:21:49'

Figure D.8: Demonstration entry for database table 'image'.

D.5 Table 'posts'

The posts entries of this table, are also referenced In this publication as “blog entries”.

Field Name	Description
pid (primary key)	The blog entry unique id.
uid	The id of the user who owns this blog entry.
headline	The headline of the entry
body	A text body.
embeddedvideos	Emved code for videos that are hosts in youtube.com or vimeo.com.
type	It can either have the 'imagesupdate' value or the 'newspost' value.
tags	A string containing the ids of all the tags the user has specified for the entry

images	A string that contains all the images of this album. It also specifies the upload order of the images,
views	A counter of how many times this entry has been viewed.
commentscounter	A counter value for all the comments written for this entry
imageview	Value that stores the way in which images are presented in each entry. The possible values are 'fullsize', 'thumbnails' and 'largethumbnails'.
visibilitystatus	Each album can either be “visible” or “invisible” to the visitors of the website.
lastupdatedtimestamp	The date and time this entry was created
creationtimestamp	The date and time this entry was last updated

Figure D.9: Structure of database table 'posts'.

Field Name	Value
pid	102
uid	5
headline	Images update #4
body	These are the images i uploaded within 24 hours!
embeddedvideos	<i>NULL</i>
type	imagesupdate
tags	0
images	446:.....:447:.....:450:.....:
views	5
commentscounter	2
imageview	fullsize
visibilitystatus	visible
lastupdatedtimestamp	'2010-08-09 15:51:28 '
creationtimestamp	'2010-08-09 15:43:48 '

Figure D.10: Demonstration entry for database table 'posts'.

D.6 Table 'tag'

Field Name	Description
tid (primary key)	The tag entry unique id.
uid	The id of the user who owns this tag.
name	The keyword specified by the user to characterize the tag.
description	A brief description of the tag.
type	Specifies what type of tag each entry is. Possible values are 'image', 'profile', 'post', 'album'.

Figure D.11: Structure of database table 'tag'.

Field Name	Value
tid	124
uid	5
name	pencils
description	image_tag
type	image

Figure D.12: Demonstration entry for database table 'tag'.

D.7 Table 'template'

Field Name	Description
tid (primary key)	The template entry unique id.
name	The name of the template
sidebarinfo	A text field containing add-on information for the websites sidebar
sidebarorientation	The sidebar of each website can have a left, right or top positioning.
imagetagcloud	Can either have the value “yes” or “no”. It specifies if an images tag cloud will appear in the sidebar of the website.
designcsscode	The tag unique id. This field is designed to store all the CSS code that is unique the each website. In the current version of the system this field is the <i>NULL</i> value.
designformoptins	The id of the user who owns this tagdesign parameters that are edited by the user, and

	are used to produce the website CSS code. The contents of this field are also needed to load the appropriate selected values in the "Design Editor" section of the website.
--	---

Figure D.13: Structure of database table 'template'.

Field Name	Description
tid	2
name	ulysses
sidebarinfo	updated:0:...:copyright:0:...:email:0
sidebarorientation	right
imagetagcloud	no
designcsscode	<i>NULL</i>
designformoptins	wallpaper:image:...:wallpapercolor:ffffff... ...:logo:text_username:...:sidebarorientati on:right:...:text:textcolor_000000:...:tex t:linkcolor_000000:...:text:linkbgcolor_ff f00:...:text:linkhovercolor_ffffff:...:text: linkhoverbgcolor_000000:...:text:navilink size_small:...:sectionsbg:trans_color_fffff f:...:text:fontfamily_Georgia, "Times New Roman", Times, serif:...:text:highlightcolor_000000:...:t ext:bghighlightcolor_ffffff:...:wallpaperty pe:empty

Figure D.14: Demonstration entry for database table 'template'.

D.8 Table 'user'

Field Name	Description
uid (primary key)	The user entry unique id
name	The name of the user
username	The username of the user (unique in the system)
password	The password of the user
email	The e-mail of the user (unique in the system)
avatar	The id of the image the user upload as a profile picture
views	A counter of how many times this user has been viewed.
tags	A string containing the ids of all the tags the user has selected to describe his/her profile

gender	'male' or 'female'
description	A brief description written by the user
facebook	The users' facebook account username
myspace	The users' myspace account username
twitter	The users' twitter account username
youtube	The users' youtube account username
vimeo	The users' vimeo account username
newsletter	Field that is not used in the current version of the system.
featured	A 'true' or 'false' value. The system currently selects the 5 most viewed users and characterizes them as 'featured'. This field will be used in a later iteration of the system
albumcomments	Specifies if the users allow comments for their albums. Possible values: 'enabled', 'disabled'
blogpostcomments	Specifies if the users allow comments for their blog entries. Possible values: 'enabled', 'disabled'.
commentnotifications	Specifies if the user want to get notified by e-mail for any new comment.
visibilitystatus	Possible values: 'visible', 'invisible'. This field can be edited by the user. If a user is invisible, then the contents of his/her website are not accessed by visitors.
accountstatus	Possible values: 'active', 'nonactive', 'hardnonactive'. This field is not used in the current version of the system. If a user the user chooses to delete his account, all the information and data that he has uploaded in the system are deleted, except from an entry in the 'user' table. These entries have the 'nonactive'. The 'hardnonactive' is a value enforced only by the system administrator.
registrationstatus	Possible values: 'complete', 'incomplete'. When a user registers to the system, then by default his/her registration status is characterized as 'incomplete' and cannot get accessed in any way, until it is changed to 'complete'.
lastupdatedtimestamp	The date and time this image entry was last edited.
registrationtimestamp	The date and time this image entry was created.

Figure D.15: Structure of database table 'user'.

Field Name	Value
uid (primary key)	5
name	james doe
username	Jamesdoe
password	dbd5ded5f0b904
email	james.doe@gmail.com
avatar	512
views	840
tags	1:.....6:.....10:.....
gender	male
description	James Doe is very cool.
facebook	jamesdoe
myspace	jamesdoe
twitter	jamesdoe
youtube	<i>NULL</i>
vimeo	<i>NULL</i>
newsletter	enabled
featured	false
albumcomments	enabled
blogpostcomments	enabled
commentnotifications	disabled
visibilitystatus	visible
accountstatus	active
registrationstatus	complete
lastupdatedtimestamp	'2010-08-13 21:03:03'
registrationtimestamp	'2010-08-13 20:54:10'

Figure D.16: Demonstration entry for database table 'user'.

D.9 Table 'usersactionlog'

Even though I believe that this is an important table for the security of the system, due to the strict time limitations of the project, and the large number of work needed in more important functions, I decided not to implement the necessary functions that will manage it. Later iterations of the system will use this table.

Field Name	Description
alid	The entry unique id.
uid	The id of the user executing the action. For unregistered users, this value is <i>NULL</i> .
message	A message describing the
entrytype	Possible values: 'routine', 'error'. For every entry marked as 'error', an e-mail is sent to the administrator of the system.
actiondatetime	Date and time the entry was created.
userip	The IP address of the user executing the action.
userprivileges	Possible values are: 'admin', 'registered' and 'common'. This field describes the user executing the action.

Figure D.17: Structure of database table 'usersactionlog'.

Field Name	Value
alid (primary key)	90
uid	5
message	Registered user logged in.
entrytype	routine
actiondatetime	'2010-08-05 19:25:24'
userip	94.*.*.3
userprivileges	registered

Figure D.18: Demonstration entry for database table 'usersactionlog'.

D.10 Table 'website'

Field Name	Description
wid (primary key)	The entry unique id
uid	The id of the user who owns this blog entry.
title	The title of the website
urltitle	The unique URL title of the website in the system. In the submitted version of the system, the username is used instead.
views	A counter of how many times this website has been viewed.
tid	The template id originally used for the design of this website.

wallpaper	Value that stores the way in which images are presented in each entry. The possible values are 'fullsize', 'thumbnails' and 'largethumbnails'.
logo	Each album can either be “visible” or “invisible” to the visitors of the website.
sidebarinfo	A text field containing add-on information for the websites sidebar.
sidebarorientation	The sidebar of each website can have a left, right or top positioning.
imagetagcloud	Can either have the value “yes” or “no”. It specifies if an images tag cloud will appear in the sidebar of the website.
designcsscode	The tag unique id. This field is designed to store all the CSS code that is unique the each website. In the current version of the system this field is the <i>NULL</i> value.
designformoptions	The id of the user who owns this tagdesign parameters that are edited by the user, and are used to produce the website CSS code. The contents of this field are also needed to load the appropriate selected values in the “Design Editor” section of the website.
uploadimageswidth	The width that every image is resized when it is uploaded. This value effects only the regular images, and not the avatar and album cover images.
uploadimagetype	Possible values '1', '2', '3'. Each value corresponds to a different setting on how the uploaded images are resized.
coverpage	The cover page of each website can either be a randomly selected image, the blog section of the website, or empty. This field stores the appropriate user selection. Possible values: 'empty', 'blogsection', 'randomimage'.
visibilitystatus	'visible' or 'invisible'. Currently not used. The visibilitystatys field in the user database is used instaid. This field was included for the case that each user could create more than one websites with the same account.
status	'active' or 'nonactive'.
lastupdatedtimestamp	The date and time this entry was created
creationtimestamp	The date and time this entry was last updated

Figure D.19: Structure of database table 'website'.

Field Name	Value
wid (primary key)	1
uid	5
title	jamesdoe portfolio
urltitle	<i>NULL</i>
views	1499
tid	2
wallpaper	wallpaper.jpg
logo	logo.png
sidebarinfo	updated:1:....copyright:0:....email:1
sidebarorientation	left
imagetagcloud	yes
designcsscode	<i>NULL</i>
designformoptions	wallpaper:image:.....:wallpapercolor:ffffff:.....:logo:text_image:.....:sidebarorientation:left:.....:text:textcolor_000000:.....:text:linkcolor_000000:.....:text:linkbgcolor_.....:text:linkhovercolor_ffffff:.....:text:linkhoverbgcolor_000000:.....:text:navilinksize_small:.....:sectionsbg:trans_white_light:.....:text:fontfamily_Georgia, "Times New Roman", Times, serif:.....:text:highlightcolor_ffffff:.....:text:bghighlightcolor_000000:.....:wallpapertype:image
uploadimageswidth	800
uploadimagetype	3
coverpage	randomimage
visibilitystatus	visible
status	active
lastupdatedtimestamp	'2010-08-14 13:23:11'
creationtimestamp	'2010-08-02 21:07:12'

Figure D.20: Demonstration entry for database table 'website'.

Appendix E: Video Files Content

Video file “./VIDEO_DEMOS/video01_main_website.mp4”.

Video that presents all the sections included in the home page of the system.

Video file “./VIDEO_DEMOS/video02_account_actions.mp4”.

Video that demonstrates:

- the registration process for a new account.
- the account activation.
- a request to reset an account password
- the password reset process
- signing-in the system
- signing-out of the system

NOTE: This video was captured when the system was operating in “localhost” mode. In this mode, all outgoing e-mails from the system are canceled, and an alert message is generated, which provides the activation code for a new account, as well as the activation code for a password reset operation. These codes are pasted in the browser address bar.

Video file “./VIDEO_DEMOS/video03_myAccount_operations_01.mp4”

Video that demonstrates the following sections of the registered users account page:

- instructions
- settings
- overview
- cover page

as well as an initial view of the generated website.

Video file “./VIDEO_DEMOS/video04_myAccount_operations_02.mp4”

Video that demonstrates the following system functions:

- creating, editing and deleting an image Album.
- uploading, deleting and editing all the relevant information of images.
- creating, editing and deleting Blog entries.
- creating a comment for an album of the generated website.

Video file “./VIDEO_DEMOS/video05_myAccount_operations_03.avi”

Video that focuses on all the available functions of the generated websites, as well as the “Design Editor” section of the users account page.

Appendix F: System Generated E-Mail Messages

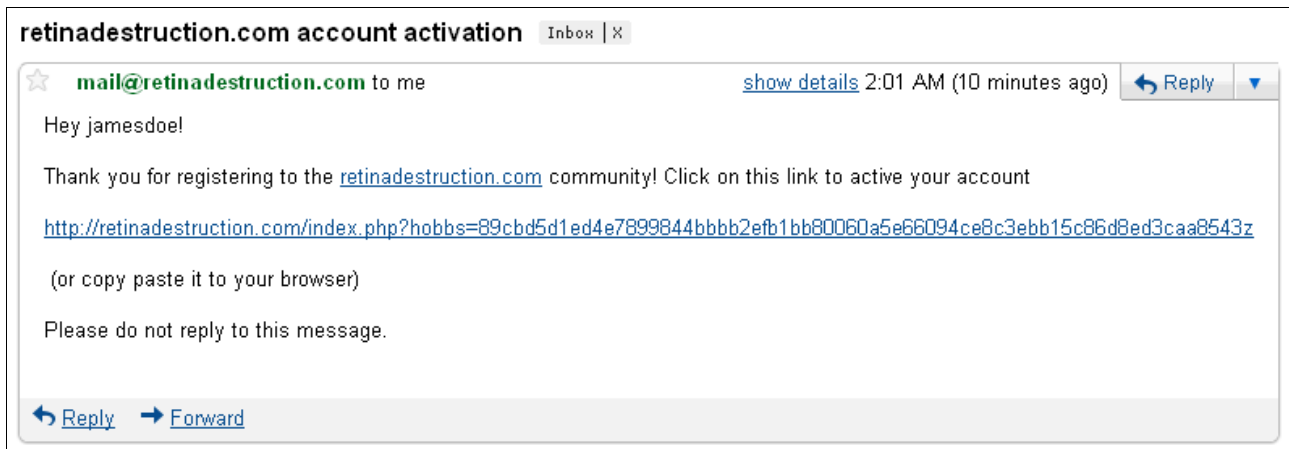


Figure F.1: Account activation e-mail for newly registered users. (screenshot taken from a Gmail account)

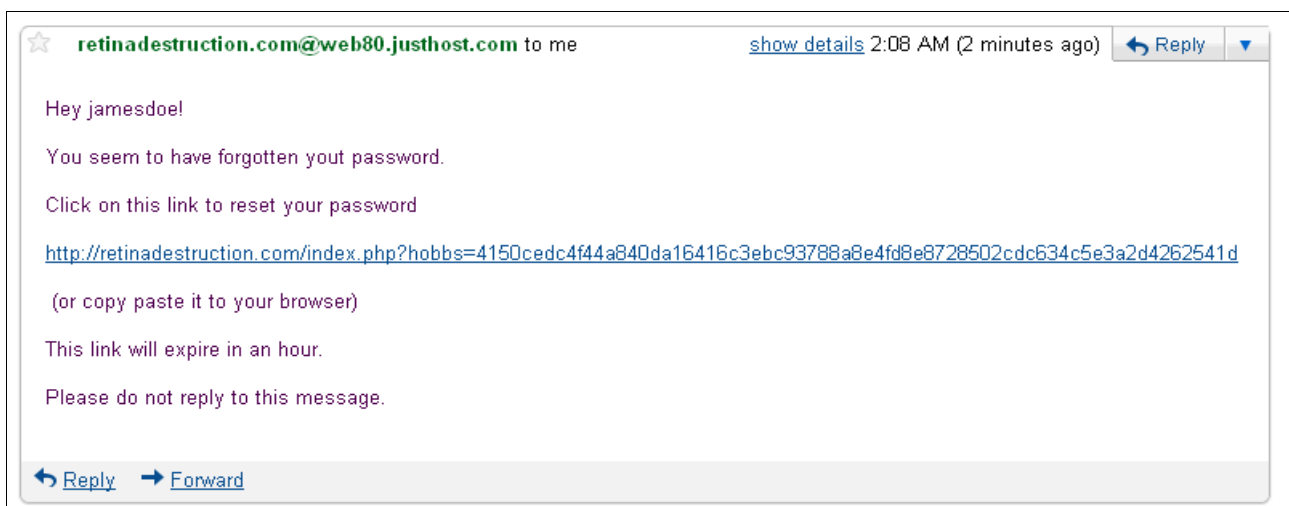


Figure F.2: Password reset e-mail, for users that filled the “forgot your password?” form (screenshot taken from a Gmail account)

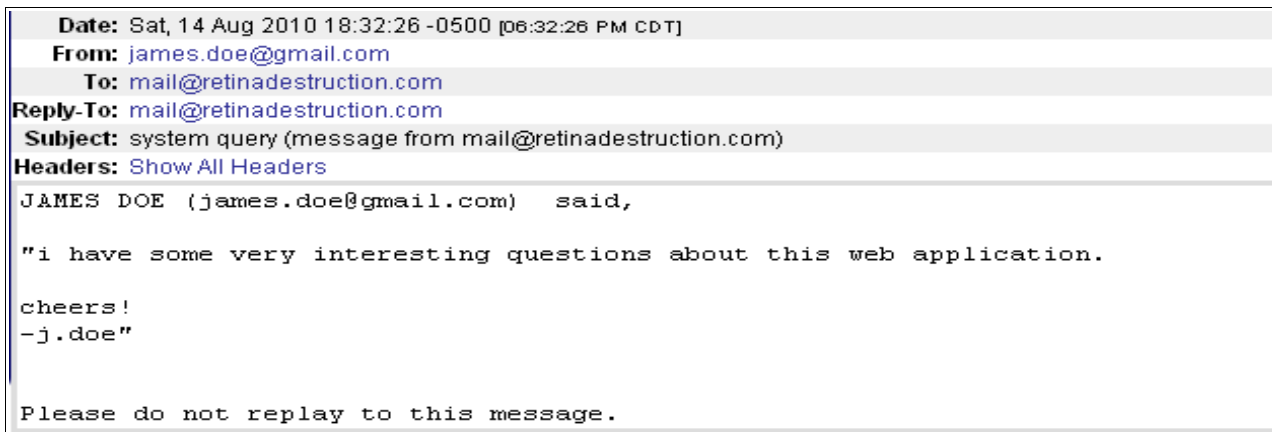


Figure F.3: E-mail to the default system account, mail@retinadestruction.com using the system's contact form.(screenshot taken from Horde webmail account).

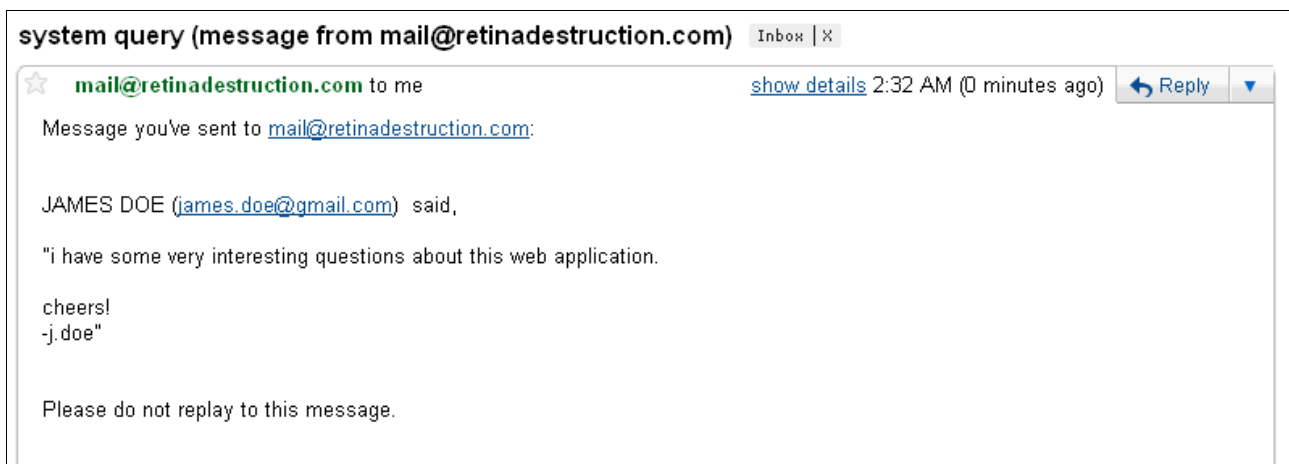


Figure F.4: E-mail to a user who sent a message to the default system, mail@retinadestruction.com using the system's contact form and selecting "Send CC to self".(screenshot taken from Gmail):

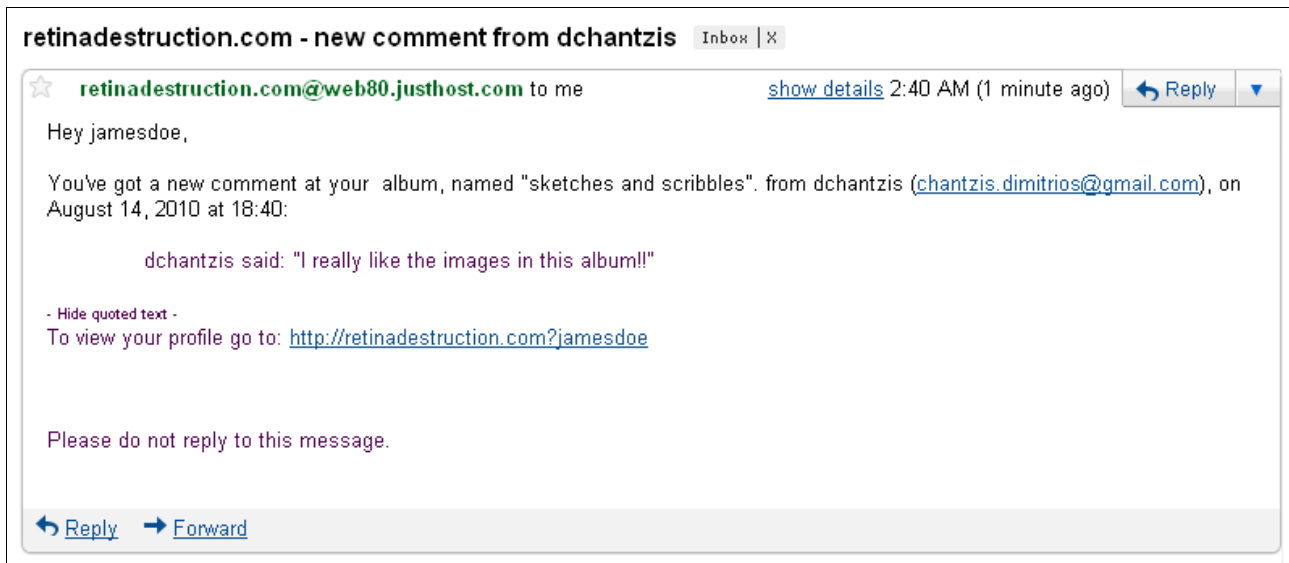


Figure F.5 Notification e-mail for new comment that was left regarding an image album of a registered user. (screenshot taken from Gmail):

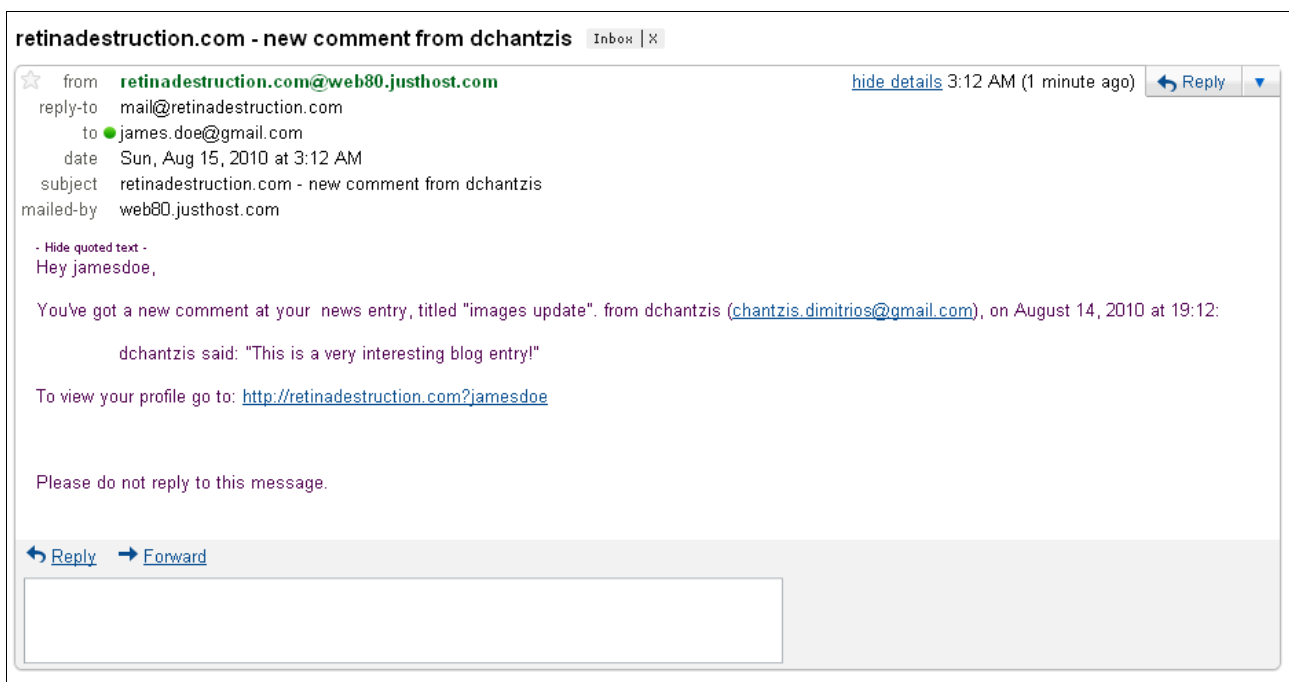






Figure F.6: Notification e-mail for new comment that was left regarding an blog entry of a registered user. (screenshot taken from Gmail):

[no subject] (message from mail@retinadestruction.com) Inbox | X




  **chantzis.dimitrios@gmail.com** to me [show details](#) 3:16 AM (0 minutes ago)  [Reply](#) 

DCHANTZIS (chantzis.dimitrios@gmail.com) said,

"hey!
i just wanted to tell you that i like your sketches!

cheers!
dimitrios"

Please do not replay to this message.

 [Reply](#)  [Forward](#)  Dimitrios is not available to chat

F.7: E-mail to a registered user of the system. This message was sent by a visitor filling the contact form from the registered user's portfolio website (screenshot from Gmail).

References

1. Newton, A., 2009, *jQuery vs MooTools*, jqueryvsmootools.com, viewed 18 August 2010, <<http://jqueryvsmootools.com/>>
2. *jQuery vs MooTools Nightly Benchmark*, n.d., blog.frontendforce.com, viewed 18 August 2010, <<http://blog.frontendforce.com/2010/05/jquery-vs-mootools-nightly-benchmark/>>
3. Retina (anatomy), n.d., britannica.com. viewed 18 August 2010, <<http://www.britannica.com/EBchecked/topic/500012/retina>>
4. Matherne B., 2002, *Art is the process of destruction*, doyletics.com, viewed 18 August <<http://www.doyletics.com/artpofd.htm> >
5. A combination of the definitions found at:
 Retina (anatomy), n.d., britannica.com. viewed 18 August 2010, <<http://www.britannica.com/EBchecked/topic/500012/retina>>
 and
 Retina, n.d., dictionary.com, viewed 18 August 2010, <<http://dictionary.reference.com/browse/retina>>
6. Destruction, n.d., dictionary.com, viewed 18 August 2010, <<http://dictionary.reference.com/browse/destruction>>
7. Nielsen, J., 2005, *Ten Usability Heuristics*, Useit.com, viewed 18 August 2010, <http://www.useit.com/papers/heuristic/heuristic_list.html>
8. magne f – the official site, n.d., magne-f.net, viewed 18 August, <<http://www.magne-f.net/1/>>
9. *Category:OWASP Top 10 for 2010*, 2010, owasp.org, viewed 18 August 2010, <http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project>
10. *OWASP Top 10 2010*, 2010, owasp.org, viewed 18 August 2010, pdf file, <<http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202010.pdf>>

Bibliography

Stroustrup, B., 2000, *The C++ Programming Language, Special Edition*, Addison Wesley

Bennet, S., Skelton J. & Lunn K., 2004, *Schaum's Outline of UML*, McGraw-Hill Professional

Ramakrishnan R. & Gehrke J., 1999, *Database Management Systems*, McGraw-Hill Higher Education

Krause, J., 2004, *Basic Design Index*, How

Andrew, R. 2004, *The CSS Anthology: 101 Essential Tips, Tricks & Hacks*, Collingwood, Australia: SitePoint

Yank, K, 2003, *Build Your Own Database-Driven Website Using PHP & MySQL*, Richmond, Australia, SitePoint

Darie, C., Brinzarea B., Chereches-Tosa, F. & Bucicia M., 2005, *AJAX and PHP: Building Responsive Web Applications*, Packt Publishing

Bulgar, B., Greenspan J. & Wall D., 2003, *MySQL/PHP Database Applications*, John Wiley & Sons

Welling, L. & Thomson, L, 2008, *PHP and MySQL Web Development*, Addison Wesley

Eisenman S., 2008, *Building Design Portfolios: Innovative Concepts for Presenting your work*, Rockport Publishers

Felton, P., 2006, *The Ten Commandments of Typography/Type Heresy: Breaking the Ten Commandments of Typography*, Merrell Publishers

Van Der Stock, A., Williams, J. & Wichers, D., 2008, *OWASP TOP 10, The Ten Most Critical Web Application Security Vulnerabilities, 2007 Update*, owasp.org, viewed 18 August 2010, <http://www.owasp.org/index.php/Top_10_2007>.

Dickinson, P., 2005, *Top 7 PHP Security Blunder*, sitepoint.com, viewed 18 August 2010, <<http://www.sitepoint.com/article/php-security-blunders>>.

Clark, D., 2004, *PHP Security Mistakes*, devshed.com, viewed 18 August 2010, <<http://www.devshed.com/c/a/PHP/PHP-Security-Mistakes/>>.

Error Handling In PHP (part 1), 2002, melonfire.com, viewed 18 August 2010, <<http://www.melonfire.com/community/columns/trog/article.php?id=120>>

Error Handling In PHP (part 2), 2002, melonfire.com, viewed 18 August 2010, <<http://www.melonfire.com/community/columns/trog/article.php?id=121>>

File And Directory Manipulation In PHP (part 1), 2003, melonfire.com, viewed 18 August 2010, <<http://www.melonfire.com/community/columns/trog/article.php?id=208>>

File And Directory Manipulation In PHP (part 2), 2003, melonfire.com, viewed 18 August 2010,
<<http://www.melonfire.com/community/columns/trog/article.php?id=211>>

Presentation, n.d., wampserver.com, viewed 18 August 2010,
<<http://www.wampserver.com/en/presentation.php>>

XAMPP for Linux, n.d., apachefriends.org, viewed 18 August 2010,
<<http://www.apachefriends.org/en/xampp-linux.html>>.

Grutzmacher, K., n.d., *Your Free MacWorld Expo Platinum Pass (valued at \$1,695)*, grutztopia.jingojango.net, viewed 18 August 2007,
<http://grutztopia.jingojango.net/2007/01/your-free-macworld-expo-platinum-pass_11.html>

Castro, E., 2006, *Bye Bye Embed*, alistapart.com, viewed 18 August 2010,
<<http://www.alistapart.com/articles/byebyembed/>>

Brown, T.B., 2003, *Creating Clean URIs With PHP*, tiffanybbrown.com, viewed 18 August 2010,
<http://tiffanybbrown.com/2003/12/22/creating_clean_uris_with_php/>

Turcsanyi, T., 2002, *mod_rewrite: A Beginner's Guide to URL Rewriting*, sitepoint.com, viewed 18 August 2010, <<http://articles.sitepoint.com/article/guide-url-rewriting>>

Kyrnin, J., n.d., *Fixed Width Layouts Versus Liquid Layouts*, about.com, viewed 18 August 2010,
<<http://webdesign.about.com/od/layout/i/aa060506.htm>>

Centering Block Element, n.d., css-discuss.incutio.com, viewed 18 August 2010,
<http://css-discuss.incutio.com/wiki/Centering_Block_Element>

Van Der Sluis, B., 2010, *Supersize that Background, Please!*, alistapart.com, viewed 18 August 2010,
<<http://www.alistapart.com/articles/supersize-that-background-please/>>

Lovitt, M., 2002, *Cross-Browser Variable Opacity with PNG: A Real Solution*, alistapart.com, viewed 18 August 2010, <<http://www.alistapart.com/articles/pngopacity/>>

Allen, D., 2001, *Reading Design*, alistapart.com, viewed 18 August 2010,
<<http://www.alistapart.com/articles/readingdesign/>>

Villarreal, S., 2004, *CSS Drop Shadows*, alistapart.com, viewed 18 August 2010,
<<http://www.alistapart.com/articles/cssdropshadows/>>

kevin, n.d., *12 Useful Tutorials to Style Web Forms*, queness.com, viewed 18 August 2010,

<<http://www.queness.com/post/1274/12-useful-tutorials-to-style-web-forms>>

Hobbs-Smith, J., 2008, *Improve your jQuery-25 excellent tips*, tvdesign.co.uk, viewed 18 August 2010, <<http://www.tvdesign.co.uk/blog/improve-your-jquery-25-excellent-tips.aspx>>

Kelly, S., 2006, *Speeding Up AJAX with JSON*, developer.com, viewed 18 August 2010, <<http://www.developer.com/lang/jscript/article.php/3596836/Speeding-Up-AJAX-with-JSON.htm>>

Green, R., n.d., *How to Write Unmaintainable Code (Humorous Article)*, freevbcode.com, viewed 18 August 2010, <<http://www.freevbcode.com/ShowCode.Asp?ID=2547>>

XSS (Cross Site Scripting) Prevention Cheat Sheet, n.d., owasp.org, viewed 18 August 2010, <[http://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)>

SQL Injection Prevention Cheat Sheet, n.d., owasp.org, viewed 18 August 2010, <http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet>

Category: OWASP Application Security Verification Standard Project, n.d., owasp.org, viewed 18 August 2010, <<http://www.owasp.org/index.php/ASVS>>

Category: OWASP Top 10 for 2010, 2010, owasp.org, viewed 18 August 2010, <http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project>

Session Management, n.d.owasp.org, viewed 18 August 2010, <http://www.owasp.org/index.php/Session_Management>

Category: OWASP Guide Project, n.d., owasp.org, viewed 18 August 2010, <http://www.owasp.org/index.php/Category:OWASP_Guide_Project>

DOM Based XSS, n.d., owasp.org, viewed 18 August 2010, <http://www.owasp.org/index.php/DOM_Based_XSS>

Klein, A., 2005, *DOM Based Cross Site Scripting or XSS of the Third Kind*, webappsec.org, viewed 18 August 2010, <<http://www.webappsec.org/projects/articles/071105.shtml>>

Output Encoding, n.d., code.google.com, viewed 18 August 2010, <http://code.google.com/p/owasp-development-guide/wiki/WebAppSecDesignGuide_D6>

Andrew, R., 2006, *Build a File Manager Using PHP and MySQL - Part 1*, Web Designer, no. 99, pp. 86-87.

Andrew, R., 2006, *Build a File Manager Using PHP and MySQL - Part 2*, Web Designer, no. 100, pp. 86-87.

Andrew, R., 2006, *Build a File Manager Using PHP and MySQL - Part 3*, Web Designer, no. 101, pp. 86-87.

Walsh, J., 2006, *Design Tips that Hit the Mark*, .net, no.153, pp. 36-40.

Oxton, J., 2006, */CSS/Create a Simple Liquid Layout*, .net, no. 153, pp. 70-72.

Oxton, J., 2006, */CSS/The Secrets of Successful Navigation*, .net, no. 154, pp. 72-74.

Grannell, C., 2006, */CSS/Layouts with Classes and Ids*, .net, no. 154, pp. 92-96.

Walsh, J., 2006, *The Colour of Web Design*, .net, no. 155, pp. 36-40.

Oxton, J., 2006, */CSS/Designing Simple, Accessible Forms*, .net, no. 155, pp. 70-72.

Lomas, R., 2006, */JavaScript/Sliders with Scriptaculous*, .net, no. 156, pp. 78-80.

Nielsen, J., 2006, *The Jakob Nielsen Experience*, .net, no. 157, pp. 38-44.

Grannell, C., 2007, */CSS/Flexible Fixed Layouts*, .net, no. 158, pp. 74-77.

Tucker, S., 2007, */JavaScript/Toggling Content Visibility*, .net, no. 160, pp. 74-77.

Grannell, C., 2007, */CSS/Move From a Tables-Based Site*, .net, no. 160, pp. 90-93.

Campbell, A., 2007, *Learn to Love Accessibility*, .net, no. 161, pp. 44-50.

Grannell, C., 2007, */CSS/Sort Out Your Drawers*, .net, no. 169, pp. 70-72.

Van Kann, P., 2007, *Secure Your Site*, .net, no. 170, pp. 38-47.

Grannell, C., 2007, */CSS/Work With Print Style Sheets*, no. 170, pp. 70-72.

Thorpe, S., 2000, *How to Think Like Einstein: Simple Ways to Break the Rules and Discover Your Hidden Genius*, Sourcebooks, Inc.

Gladwell, M., 2007, *Blink: The Power of Thinking Without Thinking*, Back Bay Books

Video Tutorials

Holzschlag, M. E. & Clarke, A., 2006, *CSS for Designers*. lynda.com, Carpinteria, CA
 <<http://www.lynda.com/home/DisplayCourse.aspx?lpk2=216>>

Weinman, B., 2010, *CSS for Developers*, lynda.com, Carpinteria, CA
 <<http://www.lynda.com/home/displaycourse.aspx?lpk2=52341>>

Hollins, C. G., 2007, *JavaScript Essential Training*, lynda.com, Carpinteria, CA
 <<http://www.lynda.com/home/displaycourse.aspx?lpk2=375>>

Skoglund, K., 2007, *PHP with MySQL Essential Training*, lynda.com, Carpinteria, CA
 <<http://www.lynda.com/home/DisplayCourse.aspx?lpk2=435>>

Skoglund, K., 2009, *PHP with MySQL Beyond the Basic*, lynda.com, Carpinteria, CA
 <<http://www.lynda.com/home/DisplayCourse.aspx?lpk2=653>>

Code Libraries

JQuery library, viewed 18 August 2010, <<http://jquery.com/>>

JQuery UI, viewed 18 August 2010, <<http://jqueryui.com/>>

Plugins/Autocomplete, n.d., docs.jquery.com, viewed 18 August 2010,
 <<http://docs.jquery.com/Plugins/Autocomplete> >

Color Picker – Jquery Plugin, n.d., viewed 18 August 2010, <<http://www.eyecon.ro/colorpicker/>>

Table Drag and Drop Jquery plugin, n.d., viewed 18 August 2010, <<http://www.isocra.com/2008/02/table-drag-and-drop-jquery-plugin/> >

SWFUpload, n.d., viewed 18 August 2010, <<http://swfupload.org/> >

jScrollPane, n.d., viewed 18 August 2010,
 <<http://www.kelvinluck.com/assets/jquery/jScrollPane/jScrollPane.html>>

Create a Vertical, Horizontal and Diagonal Sliding Content Website with jQuery, viewed 18 August 2010,
 <<http://www.queness.com/post/356/create-a-vertical-horizontal-and-diagonal-sliding-content-website-with-jquery> >

Useful Websites

<http://www.php.net>

<http://www.alistapart.com>

<http://www.apache.org>

<http://www.mysql.com>

<http://www.apachefriends.org/en/xampp-windows.html>

<http://www.wampserver.com/en/>

http://www.owasp.org/index.php/Main_Page

<http://www.lynda.com>

<http://www.sitepoint.com>

<http://www.melonfire.com>

<http://www.w3schools.com>