fancentr.org

Developing a fansite directory social network

Student name: **Sunnefa Thorarinsdottir**
Student number: **14937**
Campus: **SAE UK – London**
Course code: **WD1111**

Submission date: **26 October 2012**

Word count: **7396**

Declaration

The work in this paper is my own work, except for text I have cited from other sources in which case the text is clearly marked as such and references are given in accordance to academic regulations. I have not submitted this paper to any academic institution other that SAE Institute.

Furthermore, the views expressed in this paper are my own and do not reflect the views of SAE Institute or the University of Middlesex or any of their faculty.

Additionally, the contents of this paper may be used by SAE Institute or Middlesex University for academic purposes only.

Acknowledgements

I would like to thank my project supervisors, Su Banda and Itai Bercovier for all the help they provided me with while making this project.

Additionally I would like to thank my classmates, the other lecturers at SAE and my family.

Preface

For many years I have been very interested in fansites. I consider myself a fan of many things and my foray into the world of web development and design started with my very own fansite about my favourite singer. I have come a long way since then and I hope to demonstrate that with this project.

Fansites are quite controversial, but I believe they are an essential part of the internet and pop culture. The majority of fansites are run by teenage girls who are passionate about a certain topic and spend a significant amount of their free time promoting that topic, without anyone ever paying them to do so.

At the same time, no one has, to my knowledge, tried to combine the world of social networking so intricately with fansites. Many fansite owners use Facebook and Twitter to promote their sites, but there does not exist a social network that is exclusively built around fansites. As a fansite aficionado, this is in my opinion lacking, which is why I have tried to create such a network in this project.

Explanations of technical terms

AJAX - stands for Asynchronous JavaScript and XML. It uses JavaScript to communicate with a server thus providing continuous data.

API - stands for Application Programming interface. It is a way for developers to allow other developers access to their features.

Browser - is a program a user uses to view web pages in their own computer. There are several browsers, including Google Chrome, Mozilla Firefox and Internet Explorer.

Classes - are blue prints for objects which are reusable throughout the code and thus require less coding.

CSS - stands for Cascading Style Sheet. It is used when styling HTML and holds various rules which tell the browser how the page looks, what colours it has, what fonts and sizes, etc.

Database schema - is the entire structure of the database, including its tables, their columns and their relationships.

HMTL - stands for Hyper Text Markup Language. It is used to structure web pages and make them readable to browsers.

JavaScript - is a scripting language that is run from the browser. It is used to add interactivity to web pages.

jQuery - is a JavaScript library which allows developers to implement advanced JavaScript programming into their projects without writing a lot of code.

PHP - stands for Hypertext Preprocessor. It is a web programming language used for creating dynamic websites.

RSS - stands for Really Simple Syndication. It is used to publish news, blog posts, etc., which can then be read by a feed reader.

Server - is a program which serves web pages based on a domain name. One of the most common ones is called Apache.

Superglobal - is a predefined variable in PHP that is available throughout the entire code without the developer having to explicitly declare them as such.

UML - stands for Unified Modelling Language. It is used to plan classes and their relationships regardless of the programming language.

XML - stands for Extensible Markup Language. It is used for example when creating RSS feeds.

Table of contents

List of figures

Abstract

This project is a BSc thesis with a practical component, written as the final major project at SAE Institute, UK, London Campus.

The written component details research and planning made to aide in the creation of the practical component and a detailed summary of how the practical component was completed.

The practical component is a web application which lists fansites in a directory, allows users to signup, communicate and share fansites and keep up to date with their favourite ones.

Introduction

A fansite is a website created by fans of something to express their devotion and convey information to other fans. They can be about books, TV shows, celebrities, movies, pretty much anything that people consider themselves to be fans of.

A fansite directory then, is a website that displays links to fansites. What I am trying to do is to create a platform where fansite enthusiasts can share their favourite fansites with others, without any limitations. That means they will not be charged, I will not discriminate between hosting services or topics and the sites will not need prior approval.

The platform will provide users with the opportunity to discover new fansites, share their favourite ones and they will have a central location where they can check for updates on their frequently visited fansites by use of the RSS feed reader feature.

Because the platform is also a social network, users will be able to view each other's profiles and send each other messages.

In this report I will detail what steps I took to create the fansite directory platform, why I made these decisions and what research I did before beginning the project. I will begin by talking about my research into existing services, both fansite directories and social networking sites. Then I will talk about research into programming practices, which helped when I started developing the project.

The detailed methodology will be explained before I show the plans I made for the project. From there I will talk about the design of the site and finally about the coding and development.

Literature review

Social networks

I looked at some different social networks including Facebook, Twitter and Goodreads. Facebook has many features and seems to be a social network for the sole purpose of being a social network and not for anything else. Users can share status updates, events, links and photos, make comments, play games and interact with one another more intimately through chat and messages (Facebook, 2012). It currently has over 1 billion users (Mark Zuckerberg, 2012).

Twitter has fewer features but still allows people to share status updates and connect via messages. Its “Follow user” feature allows users to only see updates from users they want to stay in touch with, it is also possible to view posts based on “hash tags” and thus see all updates about specific content. Again, Twitter seems to be a social network designed to keep in touch with other people rather than anything else (Twitter, 2012).

Goodreads is drastically different from both Facebook and Twitter in that it does not have any status updates and it revolves exclusively around books. Users can share their favourite books, rate and review books and build lists of books they have read or want to read. Rather than just connecting users so they can communicate, Goodreads connects users so they can share books with each other (Goodreads Inc, 2012).

Similar services

I looked at several different fansite directories and concluded that one of their biggest flaws was that the listing of a site was conditional in some other way than just requiring it to be a fansite. Some of the directories required the sites to be hosted with a specific host (Fan Sites Network, 2012; Starszz.com, 2012), other’s required the sites to be about a specific topic (Pokemon Fansite Directory || The Pokemon Desert Shadow, no date) or to “pay” for the listing by displaying a link to the directory (| Celeb Listed | Directory of quality fansites...Best guide for all celebrity fansites on the net with quality content., no date).

All these directories also require admin approval of sites before they are added. This works well to ensure that the sites are of the right type or have “paid” for the listing but I want to give the user a more immediate result. Both Starszz.com and the Fan Sites Network allow the user to enter the title and description themselves when they suggest a site. This results in some sites having no description at all or a title that is nothing like the title tag on their site. I want

to extract this information directly from the site because I believe it will provide my users with more accurate information.

Although both Starszz.com and the Fan Sites Network offer the users to enter an RSS feed URL in their suggestions, I was unable to find any place where they use that information. I want to let users use the sites' RSS feeds to find out if there are any new updates.

Programming practices

I looked at several books and websites to help me understand more about PHP programming and development. Because the books disagreed with what I learned in lectures at SAE on some matters, I will mention what my lecturer had to say as well.

The Model View Controller pattern makes managing code easier by separating the content from the presentation and the logic. When using PHP, not all sources agree on wether the controllers should be classes or scripts. In one of my lectures at SAE, I was told by the lecturer that the controllers could be either simple scripts that mostly operated on switches or "if" statements or classes (Banda, 2012). One of the books I read recommended that controllers should only be classes (Peacock, 2010, p. 23). I decided to take the script approach because I would not have to create a second controller to instantiate and work with the controller objects, even though there would be more code in that one script.

Both Peacock and Banda recommended using a database abstraction layer, meaning that an object should be created with the sole purpose of communicating with the database. The models would then require an instance of this object (Peacock, 2010, p. 23).

Peacock uses the factory pattern in his book to create objects that all aspects of his framework need. This includes the database layer, template parsing and URL processing. This means that those helper objects are never accessed directly, only through the factory (Peacock, 2010, pp. 28 - 31). I fail to understand why it is done this way and so I have not implemented this approach. The factory pattern is perhaps useful when the code requires many helper objects that need to be available everywhere, but that is not the case in my code.

Peacock also mentions the front controller pattern, something that I have used in the past without knowing that's what it is called. He explains that sending all requests through one "index.php" file eliminates the need to include the same files over and over again and copying and pasting code in different places is no longer necessary (Peacock, 2010, pp. 56 - 57). Matt Zandstra, author of *PHP Object, Patterns and Practice*, explains the front controller pattern in a completely different way. Peacock explains the concept and offers short ex-

amples bu Zandstra offers more complex sample code and suggests creating separate objects for requests and commands, without showing what code should be in the “index.php” file (Zandstra, 2010, pp. 235 - 245).

Peacock also explains the use of “.htaccess”, a file that can be used to pass certain directives to the Apache web server, to aid in the front controller pattern. This is also something I have done before, although Peacock’s approach is slightly different to what I was taught at SAE. Peacock saves the entire query string in the same element of the $_GET superglobal, then uses a class to deconstruct it (Peacock, 2010, p. 58). Banda was against this approach because it means using numbers to keep track of the query strings instead of saving each part in a different $_GET element with a unique name (Banda, 2012).

I have tried both approaches in projects before and I must say that while the first approach makes the “.htaccess” file quite neat and tidy, keeping track of the URL pieces can be quite difficult at times. The other approach, writing separate rules for every possible scenario, separating the query strings and giving them each a unique name, takes more detailed planning in the beginning but makes the code easier to understand.

Zandstra talks very much about design patterns in his book. At the same time as he says to not use a pattern, or even objects, when it is not necessary, I could not help feeling like he was creating an object for absolutely everything in his samples, sometimes just for the sake of having an object.

He explains the patterns in the book quite well and as I was reading through them I realised that I had used the concepts of some of these patterns before (front controller, singleton, template view) though I had not implemented them in the same way as he had (Zandstra, 2010).

One of the best sources I have found to learn about PHP is the PHP online manual. It takes a little while to understand how to use it but it provides detailed information about every functionality that is available, usually coupled with examples from both the authors of the manual and its users (PHP Documentation Group, 2012).

In the manual is where I found the most detailed yet simple examples of the use of the cURL extension, which is used to connect to a remote server through a script. The manual explains what is needed for it to work and gives several different examples based on what the user might want to get from using it. In order to use cURL a few lines of code are needed, all of which do different things which are necessary to build a request. It is possible to specify if a response from the remote server is required, what data to send to the server and what user agent the script presents itself as. The recommendation in the manu-

al is that the script pretend to be a browser so there is more chance that it won't be blocked (PHP Documentation Group, 2012).

Another very useful manual that I looked at was the jQuery manual. It gives detailed explanations and examples of how to use every aspect of jQuery, both how you can integrate it into your HTML and how it works (The jQuery Project, 2010). One of the most useful functionalities for this project found therein is the shorthand AJAX function, which allows you to make an AJAX request in one line (The jQuery Project, 2010).

Methodology

I began the project by making detailed plans. I decided what features I wanted to implement based on what features other fansite directories have. Then I mapped out the project by creating activity diagrams for each feature. Following that I created the database schema and decided exactly what information to store.

This then lead me to create class UML diagrams, first ones based on the database schema I had already created, then additional ones that did not relate to any database.

After the planning stages I followed the plans when programming the project. As sometimes happens when programming starts, aspects of the plans were not put into effect or were changed. During the programming stage, I conducted research into various subjects relating to the programming such as private messaging systems and APIs for displaying screenshots of websites. Doing the research in parallel with the programming helped me a lot more than if I had done the research first since it allowed me to test out theories and pick out the best approaches as I went along.

After each feature was fully programmed, I tested it thoroughly and tried to account for every possible scenario. This sometimes resulted in the need to go back and change a large part of the programming due to circumstances I had not foreseen when writing the code.

When I had finished the programming for each feature I mapped out its HTML structure. Because I programmed the project in such a way that the HTML could be easily changed without affecting the rest of the code, I was able to finish the programming and testing with a temporary layout.

After the programming was finished I developed a few wireframes and three detailed Photoshop mockups. I then converted one of the mockups into a fully functional layout, integrating it into the HTML structure I had already created.

During the design stage or testing of other features I sometimes noticed something I had forgotten to implement fully or change. This made me look over my code several times during the programming and design stages, even though I had already finished work on a specific feature.

Project planning stage

Before development starts it is always a good idea to have a plan. A good plan can make development easier and less time consuming. I started by looking at existing fansite directories and building a list of features I wanted to have. From there I created activity diagrams for each feature, UMLs for each class I planned to have and the database schema, and decided how the database tables related to each other.

Feature list

The features are based on what other fansite directories have and how to improve on what they have done. An analysis of their features were detailed in the Literature Review chapter. In at least two cases, users had to be logged in to access all the features but could view a list of sites without being logged in.

User registration and login system

The user registration and login system should be simple, it's main purpose is to allow users to interact and to keep track of which user adds which site and owns which RSS feed list.

A site directory

The site directory should allow users to view a list of sites in different order based on the alphabet, the popularity of the sites and the date they were added. Additionally users should be able to add new sites to the database. The adding of new sites should be done in such a way that the user does not have any control over what information is entered into the database. That information must come directly from the site. This will require a program that will extract the information from the site's HTML source. For legal reasons, the program must also check for permission to access the site in such a way.

A screenshot of each site should be displayed next to the site's description and title. This will require the use of a third party service.

RSS reader

Users should have the ability to create lists of sites they wish to view the RSS feeds from. The RSS feeds should display post titles from the last 7 days with links to the sites. Users should also be able to edit and delete these lists.

Private messaging between users

Users should be able to send each other private messages. Unread messages should be highlighted and the navigation link to the messages page

should indicate if the user has a new message or not. Users should also be able to delete old messages.

Modular page setup

The pages should be made up of several modules. The modules required are:

Header

The header module should include a login form and show the user's display and message link if they are logged in.

Footer

The footer module should show a list of links to text-only pages such as "Terms of Use" and "Privacy Policy".

Sites

The sites module should be responsible for the site directory, including listing sites and adding sites.

Users

The users module should take care of the users, including logins, registrations and profiles.

Ads

The ads module should load a single, randomly selected ad code from the database.

RSS feeds

The RSS feeds module should handle viewing, creating, editing and deleting feed lists.

Navigation

The navigation module should show the main navigation links, excluding the pages linked in the footer. It needs to show a different list of links depending on whether a user is logged in or not.

Text

The text module should load the correct text associated with a page from the database based on that page's name.

Contact

The contact module should show a contact form and handle the sending of that form to a specified email address.

Welcome message

The welcome message module should show a welcome message, which is a common practice in traditional fansites.

Statistics

The statistics module should show simple statistics about the sites and users as well as a link to the last added site.

Messages

The messages module should deal with private messages, including reading and sending them.

Each module should be, as much as possible, developed independently of other modules. Modules which require other modules to be completed first, such as the messages module, should be completed last. Each module should only appear once on each page and some modules, such as the sites and messages, must not appear on the same page.

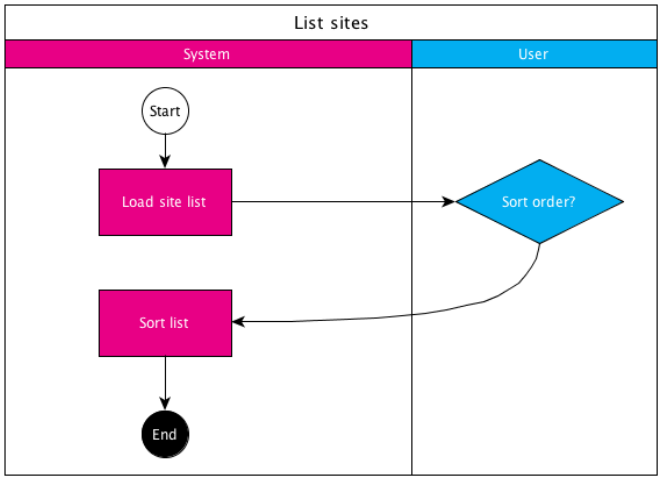
Template parsing system

To make creating different layouts easier, HTML and PHP should be mixed as little as possible. Instead, a class should be developed which will load an HTML source and replace specific token tags with information loaded through PHP code. This will make it easier to change the HMTL structure without needing to change the PHP code.

Activity diagrams

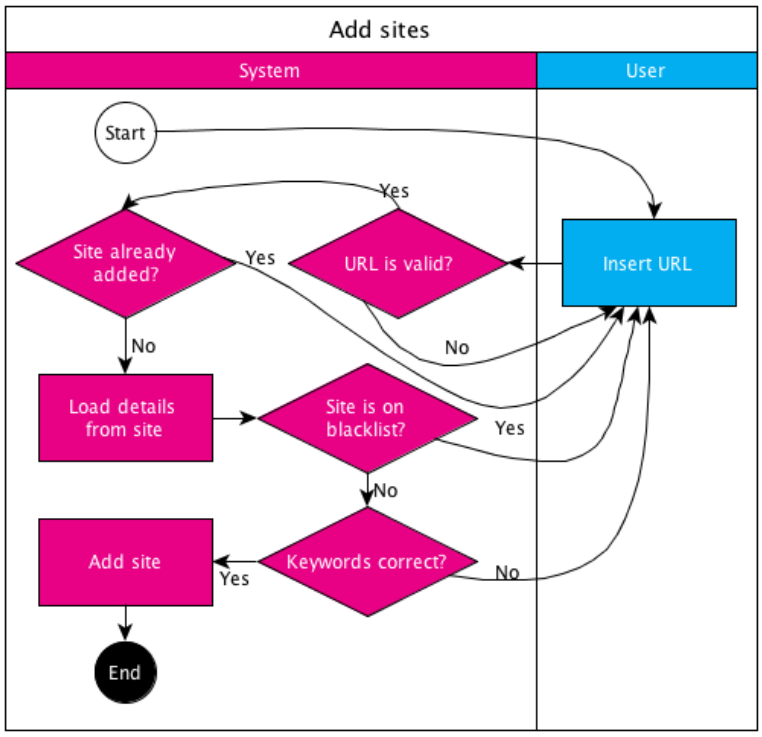
Each of the features have multiple steps. To better understand these steps and in what order they should be I created activity diagrams. These diagrams show the logic of the feature and will help in creating the code that will make them work.

List sites activity



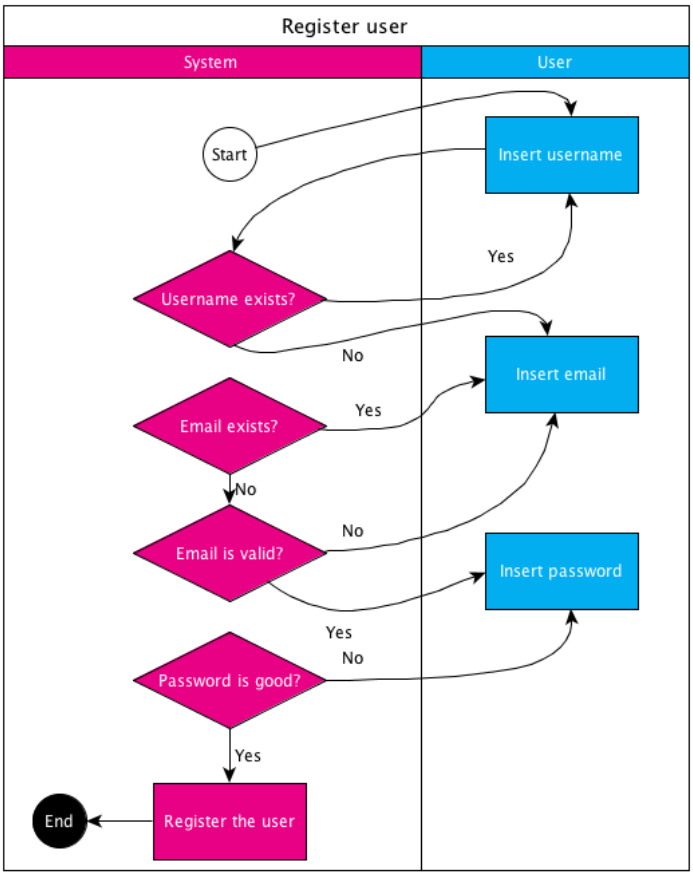
The list sites activity starts by loading a list of sites, then sorts the list according to the user's preference.

Add sites activity



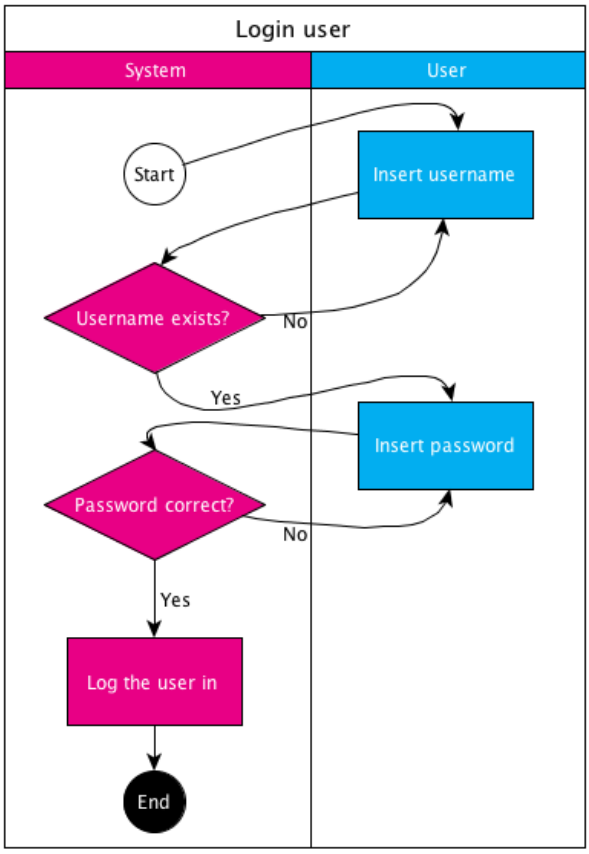
The add site activity starts when the user inputs a URL, then it tests that URL and asks for a new one if any of the tests fail. Only if the URL passes all tests is it added to the database.

Register user activity



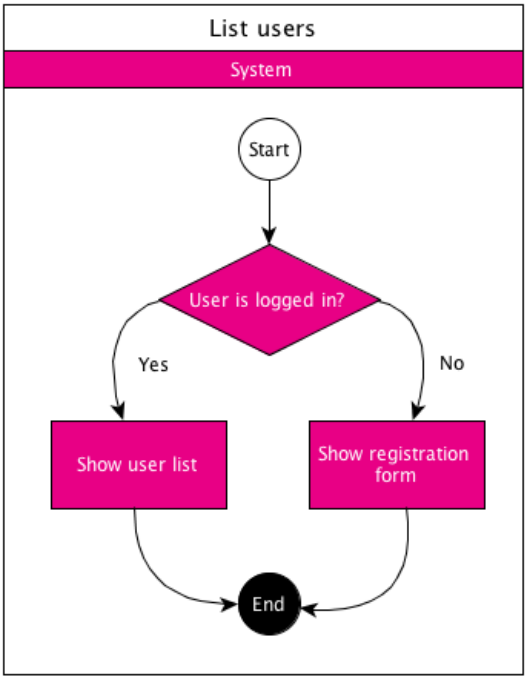
The register user activity starts when the user inserts their desired username, email address and password. Several tests are performed to ensure the validity of the data and only if the data passes all tests is the user added to the database.

Login user activity



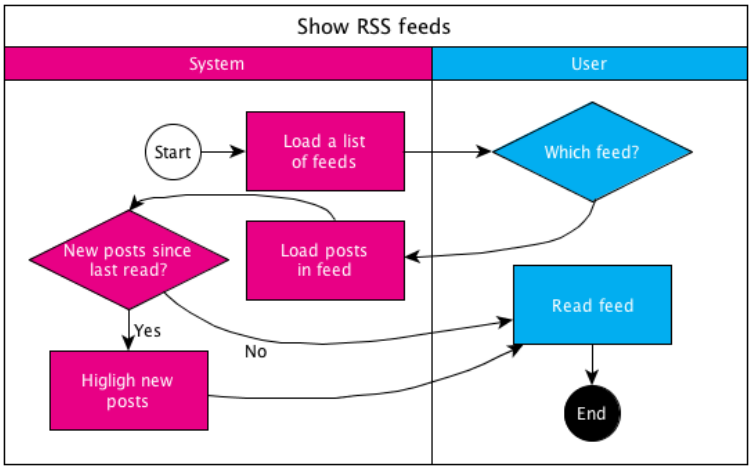
The login activity starts with the user inputting their username and password, then it checks those are valid and only logs the user in if they are.

List users activity



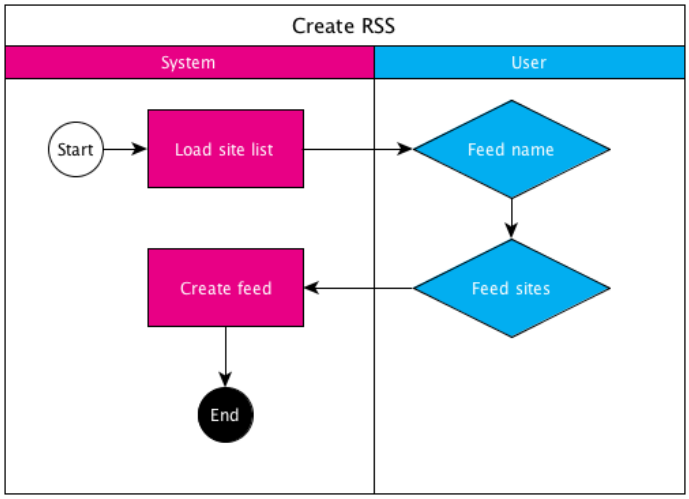
The list users activity starts by checking if a user is logged in. If no one is logged in, a registration form is shown, otherwise the user list is shown.

Show RSS feeds activity



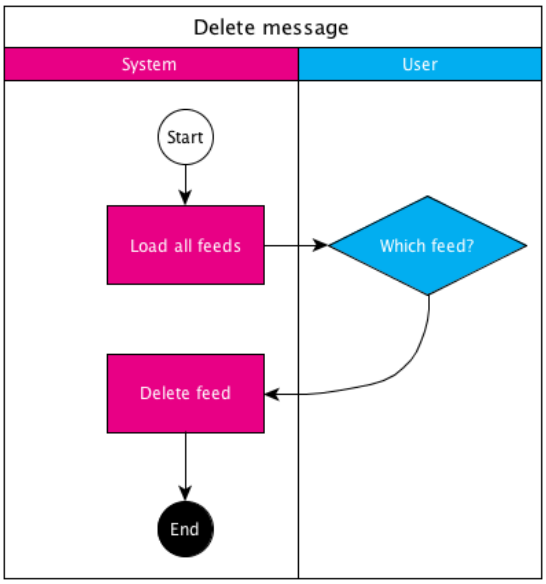
The show RSS feed activity loads a list of feeds, then loads the posts in the feed chosen by the user.

Create RSS activity



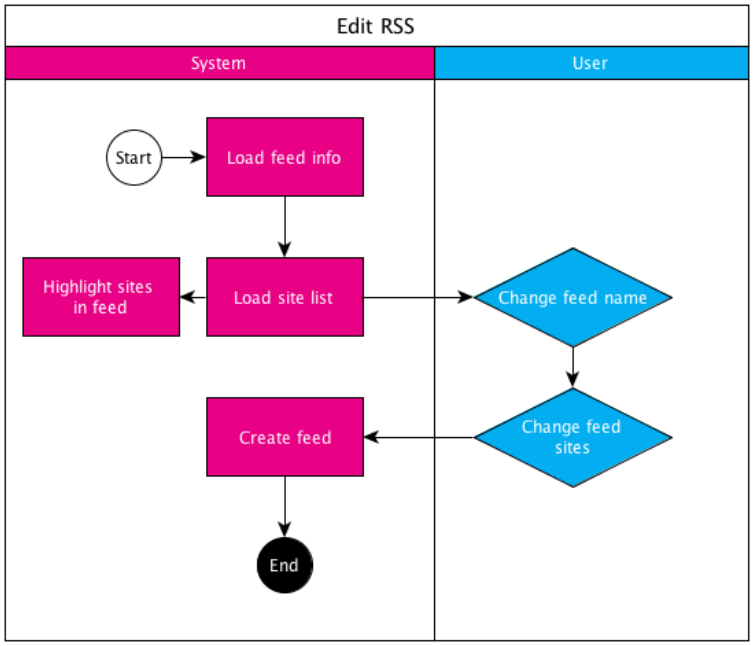
The create RSS feed activity starts by loading a list of all RSS ready sites. The user then chooses the feed name and what sites they want and the feed is created.

Delete RSS activity



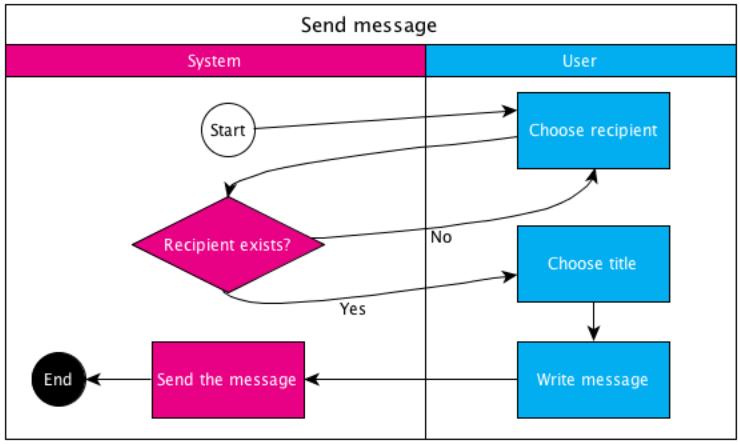
The delete RSS feed activity starts by loading a list of feeds. The user then decides which one to delete and it is deleted.

Edit RSS activity



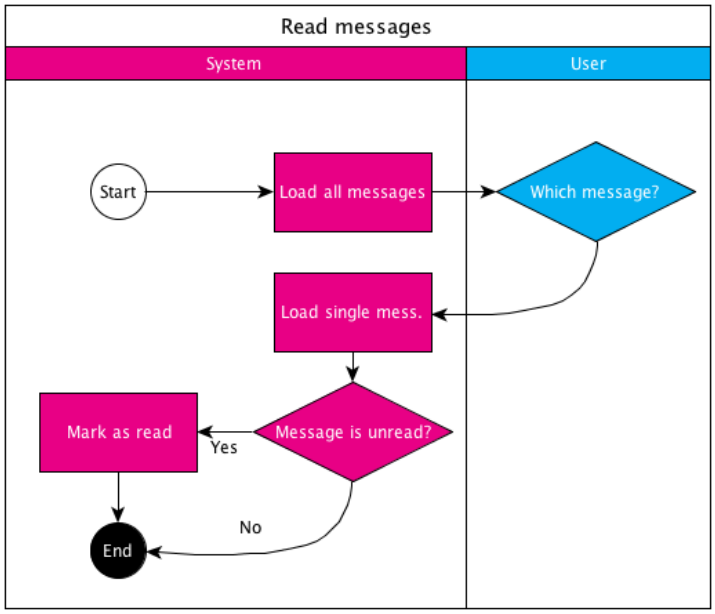
The edit RSS feed activity starts by loading the feed info and the site list, and highlights which sites are in the feed. The user then chooses if they want to change the feed name or sites and the feed is updated.

Send message activity



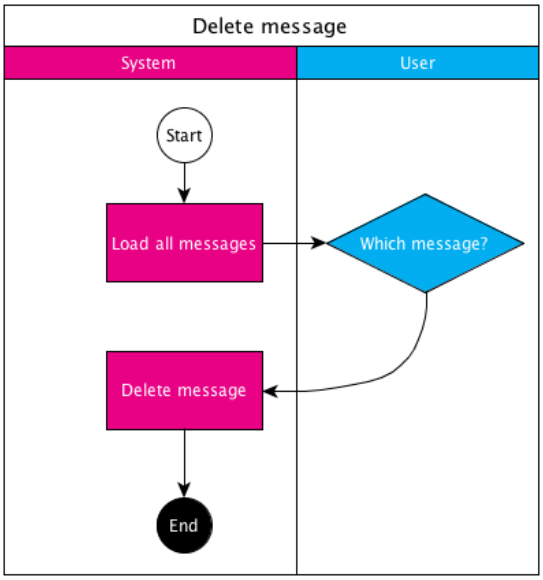
The send message activity starts with the user choosing a recipient. If the recipient does not exist the user must choose a different one, otherwise they can write a title and compose a message.

Read messages activity



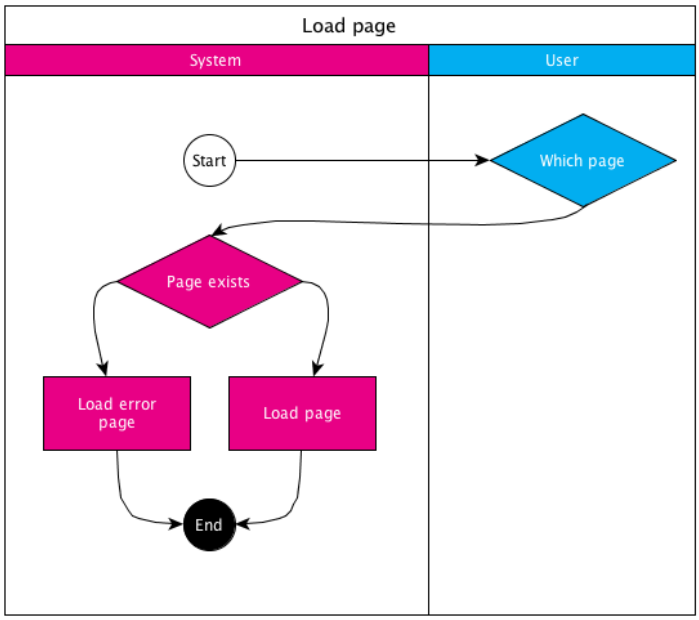
The read messages activity starts by loading all messages. The user chooses what message to read, which is then loaded. If that message was previously unread the message is marked as read.

Delete message activity



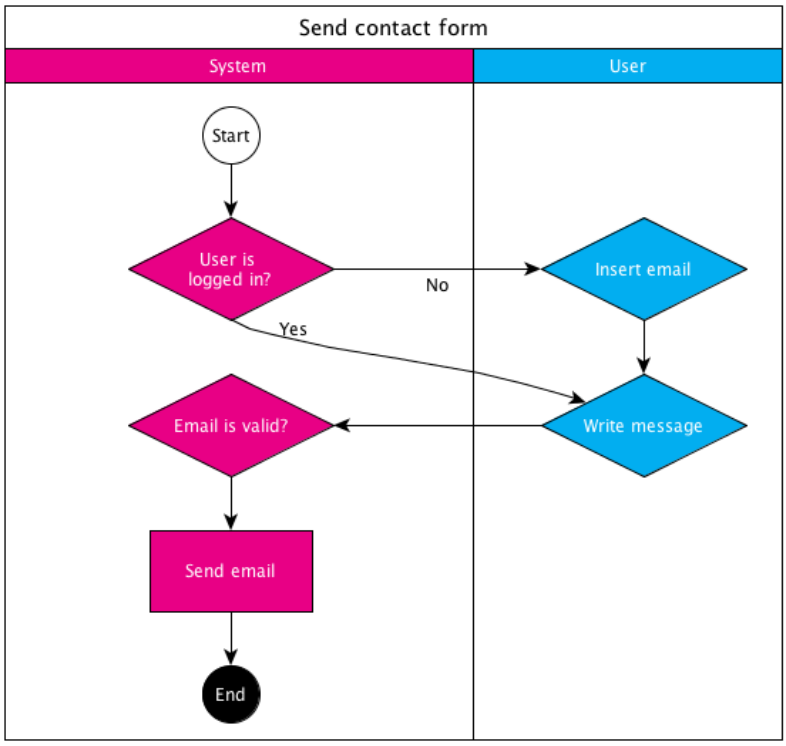
The delete message activity loads a list of messages and the user chooses which one to delete.

Load page activity



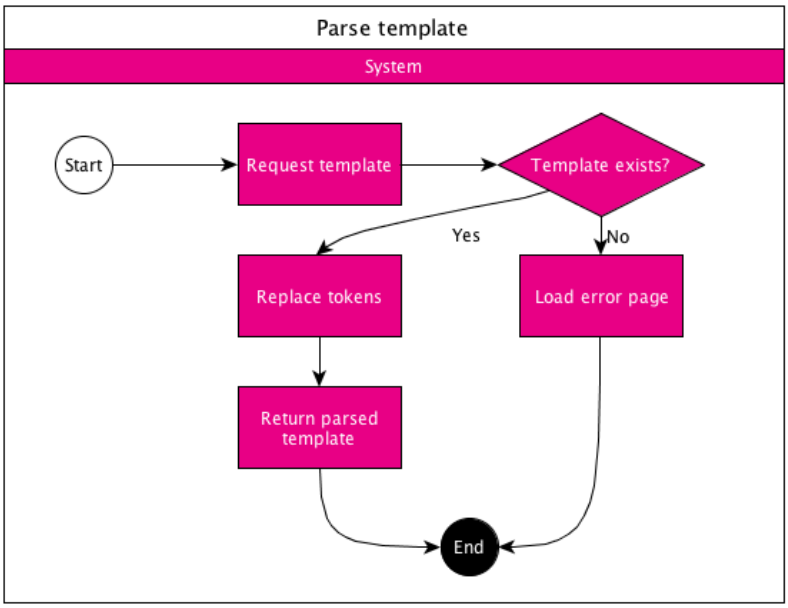
The load page activity starts with the user choosing which page to load. If that page exists it is loaded. If not, an error page is loaded.

Send contact form activity



The send contact form starts by checking if a user is logged in. If they are, they can compose a message which will be shown as being sent from them. If no one is logged in, the user must insert their email address and then compose a message.

Parse template activity

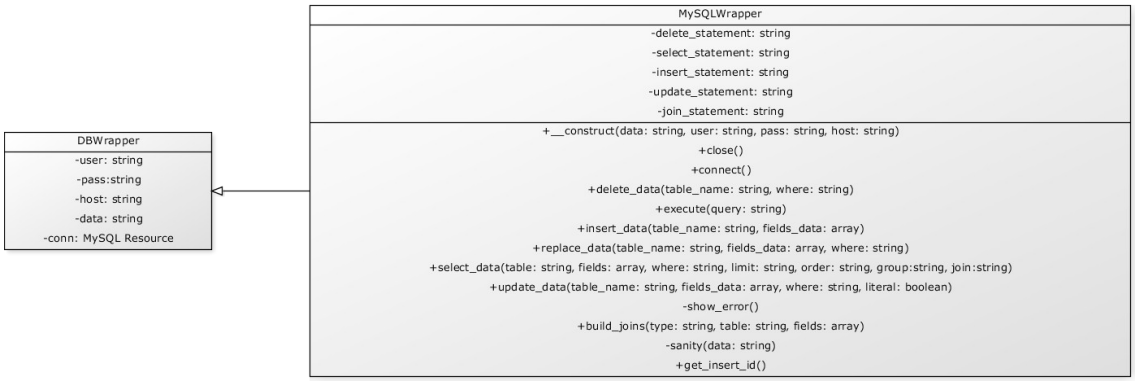


The parse template activity starts by requesting a template and checking if it exists. If it does, the template will be parsed, if it does not an error page is loaded.

UML diagrams

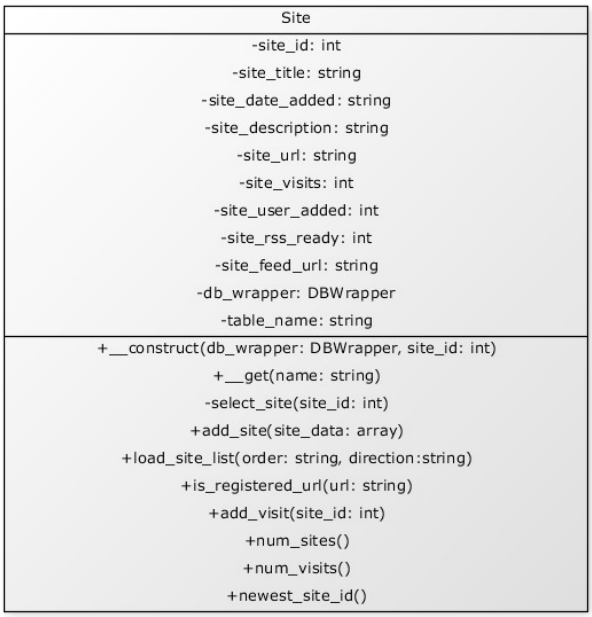
Creating UML diagrams and showing how each planned class relates to the others can significantly cut down on development time. By planning out what properties and methods a class needs, all that is left to do is to implement them.

Database wrapper



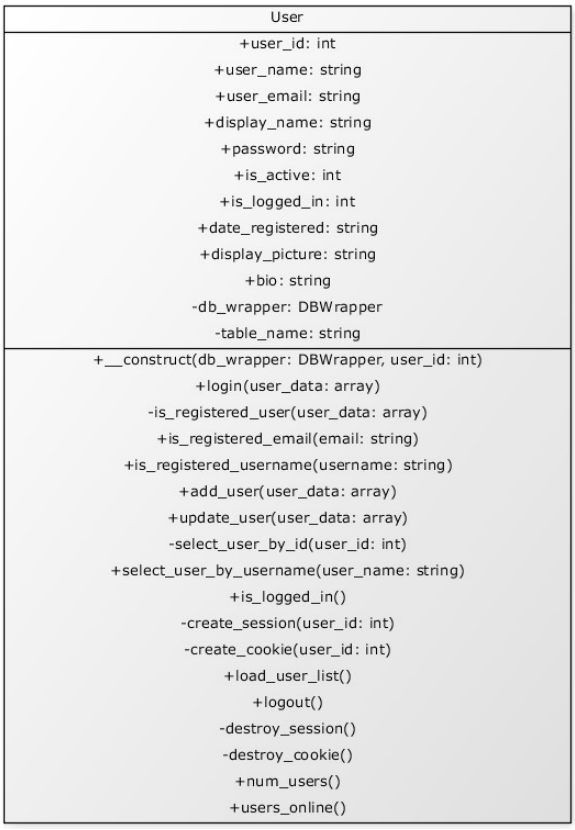
The DBWrapper object is an abstract class. MySQLWrapper inherits all its methods from there.

Site



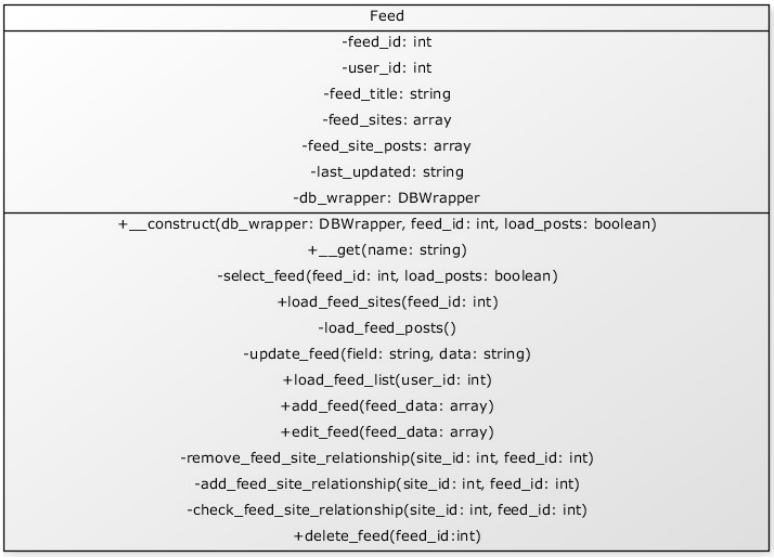
The site object represents a single site from the database, and should also be able to return a list of sites. It requires an instance of DBWrapper.

User



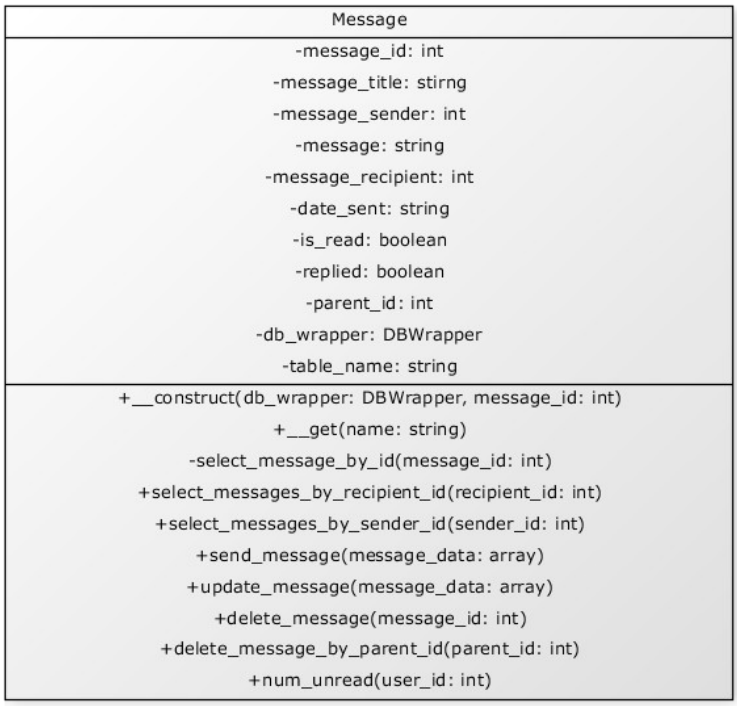
The User object represents a single user, but should also load a list of all users. It requires an instance of DBWrapper.

Feed



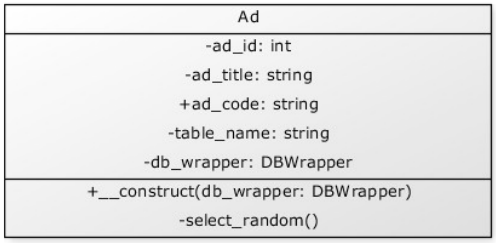
The Feed object represents a single feed list and should be able to load all feed lists by a given user. It requires an instance of DBWrapper.

Message



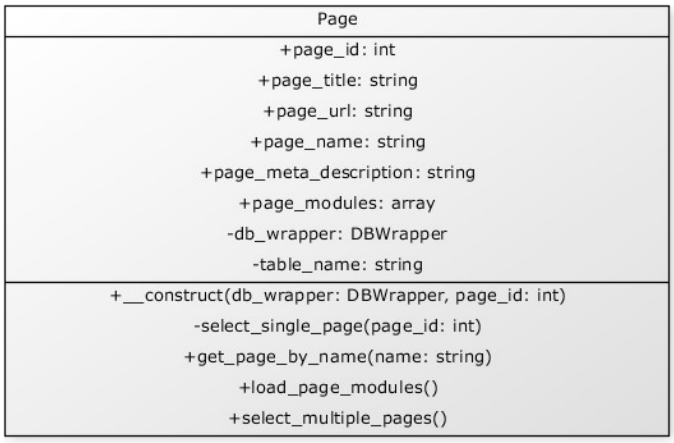
The Message object represents a single message and can load a list of messages based on a specific user id. It requires an instance of DBWrapper.

Ad



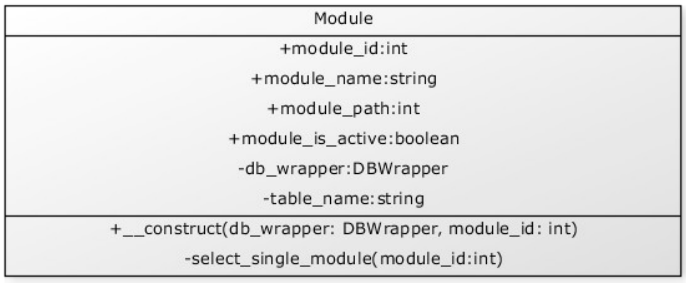
The Ad object represents a single ad code. It requires an instance of DB-Wrapper.

Page



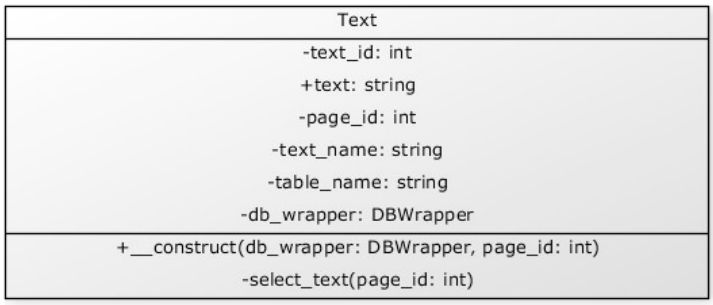
The Page object represents a single page, but also loads a list of pages and all the modules associated with that page. It requires an instance of DB-Wrapper.

Module



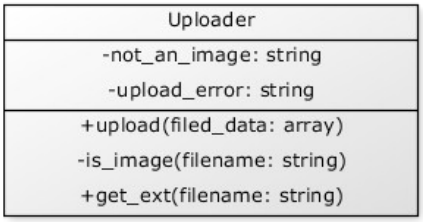
The Module object represents a single module. It requires an instance of DBWrapper.

Text



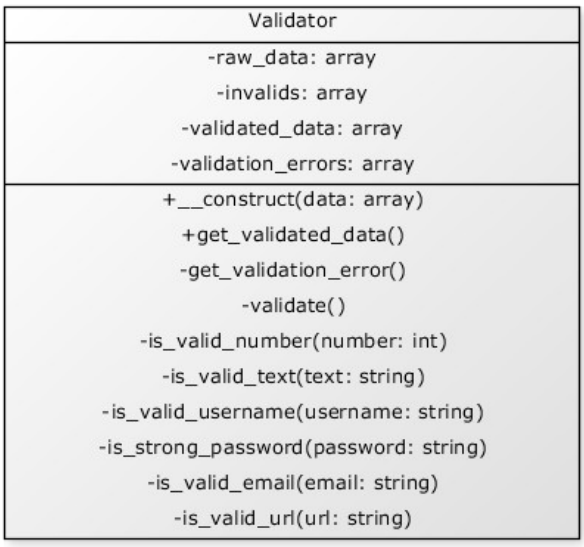
The Text object represents a single text. It requires an instance of DB-Wrapper.

Uploader



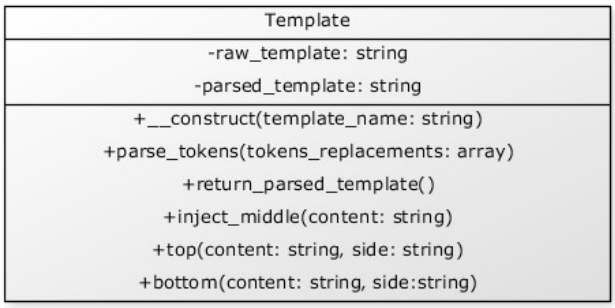
The Uploader object uploads a single image.

Validator



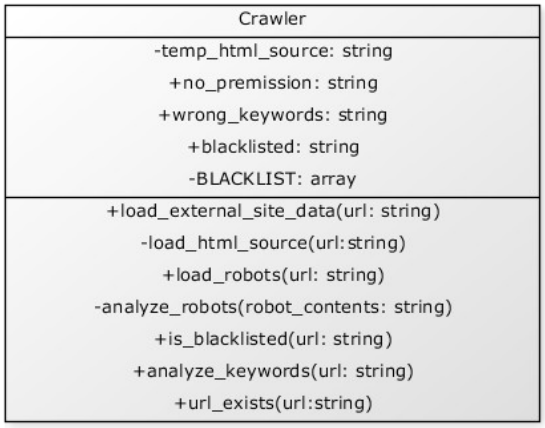
The Validator object validates an array of data and returns either the data or an error.

Template



The Template object loads an HTML template and replaces its tokens with data.

Crawler

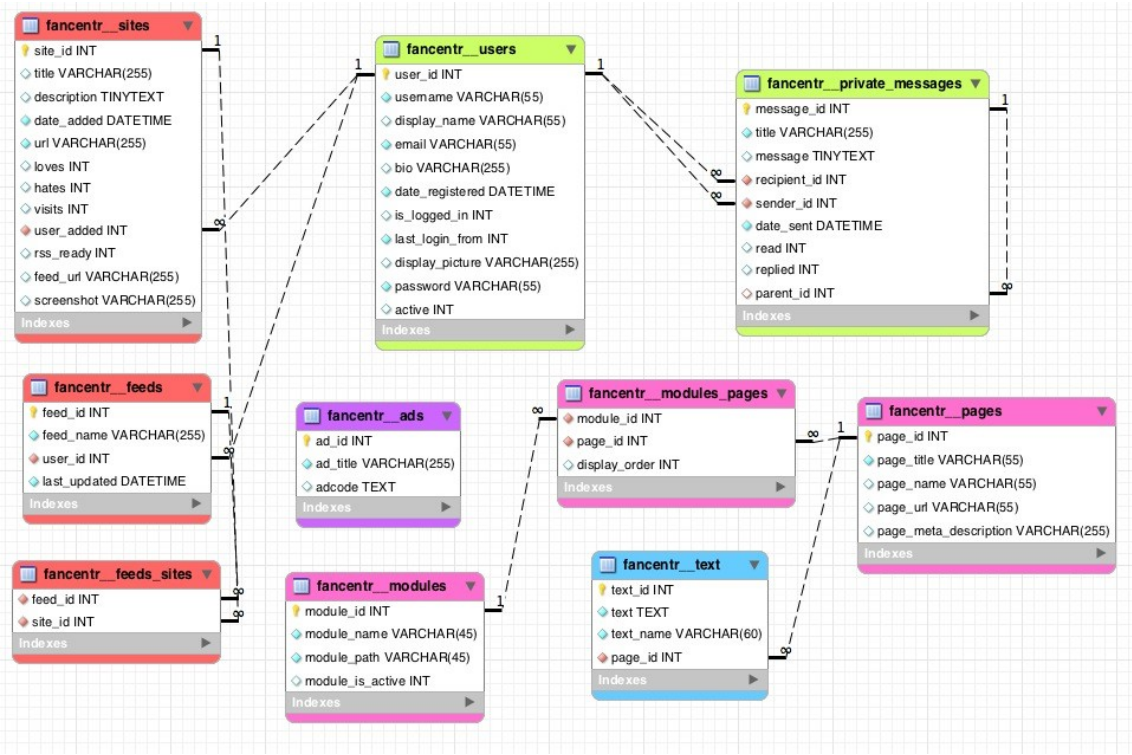


The Crawler object crawls a website and mines its HTML source for data.

Entity relationship diagrams

Deciding what information should be in the database makes programming easier because the database is already prepared and not being developed at the same time as the code.

Database schema



Project design stage

The colours and imagery are very simple. This is because the design is based more on social networking sites that fansites or fansite directories. All the directories examined had some sort of imagery reminiscent of fansites, at the very least they had some images of celebrities or used fonts and brushes common in fansite layouts. Most of the images on the site are screenshots of the sites that are listed. The only other images are decorative icons and a logo.

There is also an ad in the layout, which is something both fansites and social networks have in common. It is a simple Google ad, but I think it makes the site appear more like it might be hosted by a fansite hosting company. It has been my experience that a lot of fansite owners do not consider a fansite worth much unless it is hosted by one of the big companies.

The appearance of the statistics module is based on the Fan Sites Network hosted sites directory which has a similar feature. The appearance of the navigation module is more based on a social network, though the icons are a bit bigger than on most of the sites I looked at (Facebook, Twitter).

I used CSS sprites for the decorative images and the logo. This is the first time I have ever fully implemented CSS sprites and I was surprised at how easy it was. Once I figured out the coordinates of each image, the CSS code was quite simple.

I created HTML pages with tokens that were later replaced by content based on the PHP code. This made it easier to keep track of what information was supposed to be where. Instead of some complex PHP code to and comments mixed in with the HTML simple keywords, or tokens, were used. That way I had “{SITE_TITLE}” to represent the site title instead of PHP code like this:

```
<?php foreach($sites as $site): ?>
<p><?php echo $site[‘title’]; ?></p>
<?php endforeach; ?>
```

While the efficiency of token replacement is debatable from a programming point of view, from designer’s perspective they make more sense.

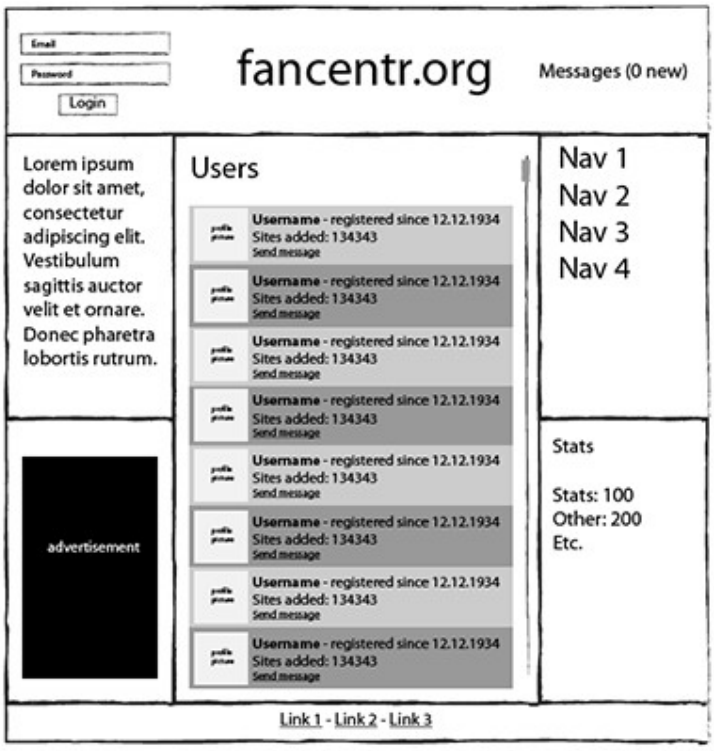
Wireframes

Before I created the final design with all the colours and imagery I created wireframes; simple drawings of where I wanted the information to be on the page. This helped when deciding what information to extract from the database and in creating the HTML structure for each page.

Home page / site list

Add site

User list

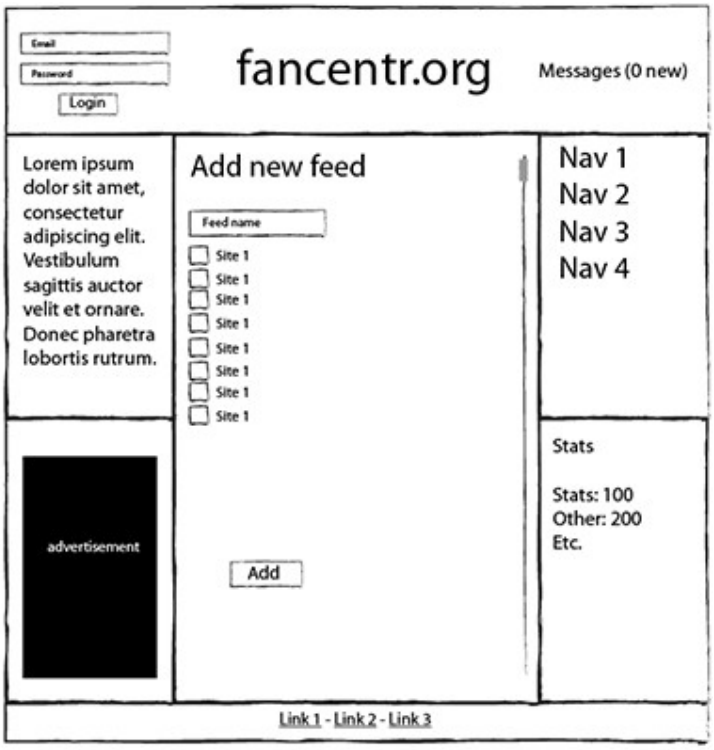


Register

Profile

RSS feed list

Create RSS feed



Read RSS feed

Messages List

Send message

Read message

Text

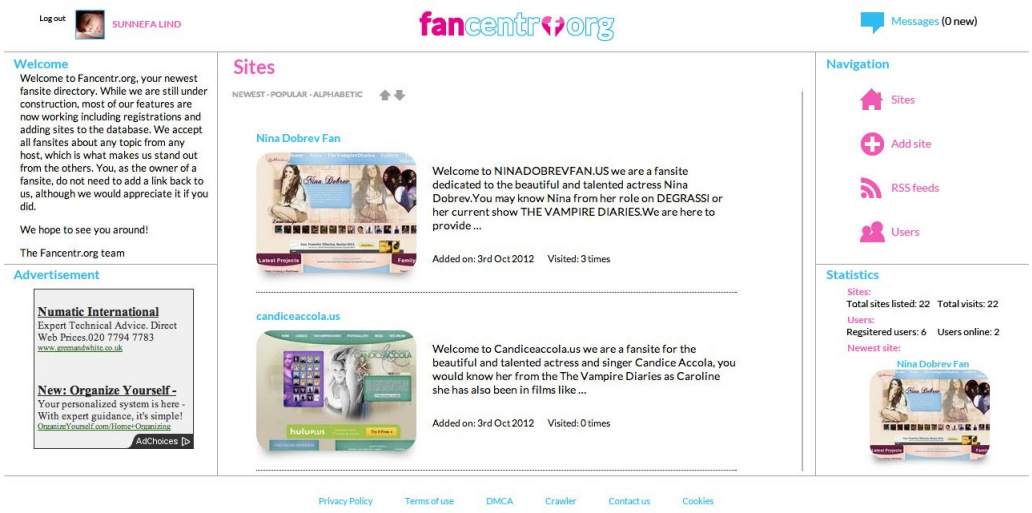
Detailed mockups

I created three detailed mockups of the home page to help me decide which one to implement. This made it easier to create a CSS stylesheet as I had already decided which colours and images to use.

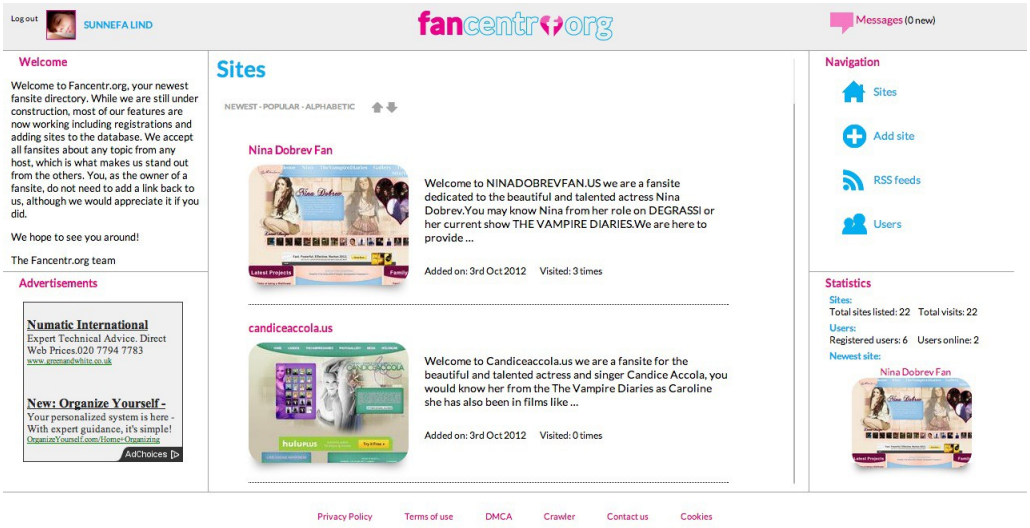
I started by creating the logo and deciding its colour scheme. I wanted to have a unique logo, both something that could be large with the whole site name and something that could be made into an icon. I based the icon on Twitter, Facebook and Tumblr icons, while the rest of the logo was inspired by Flickr's logo.

The colour scheme for all three mockups is based on the logo's colour scheme.

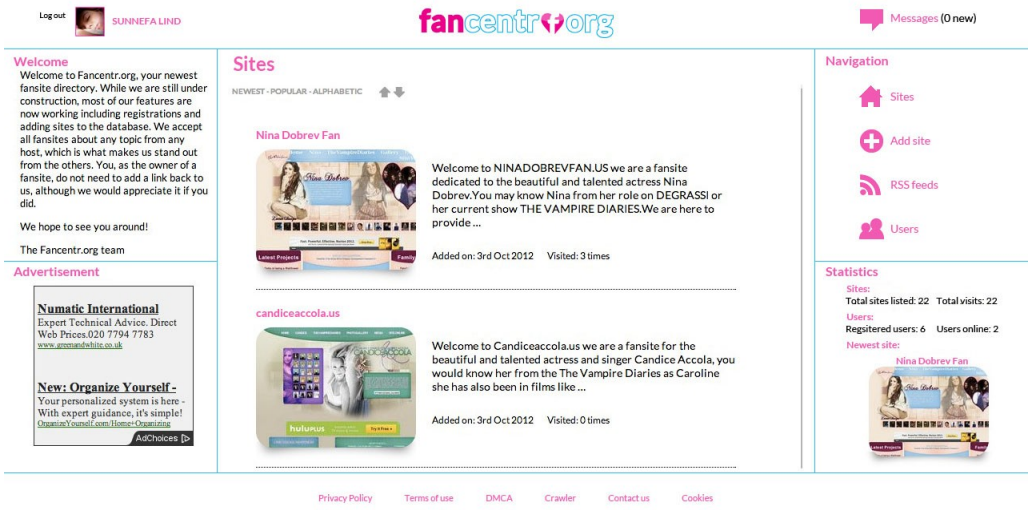
Mockup 1



Mockup 2



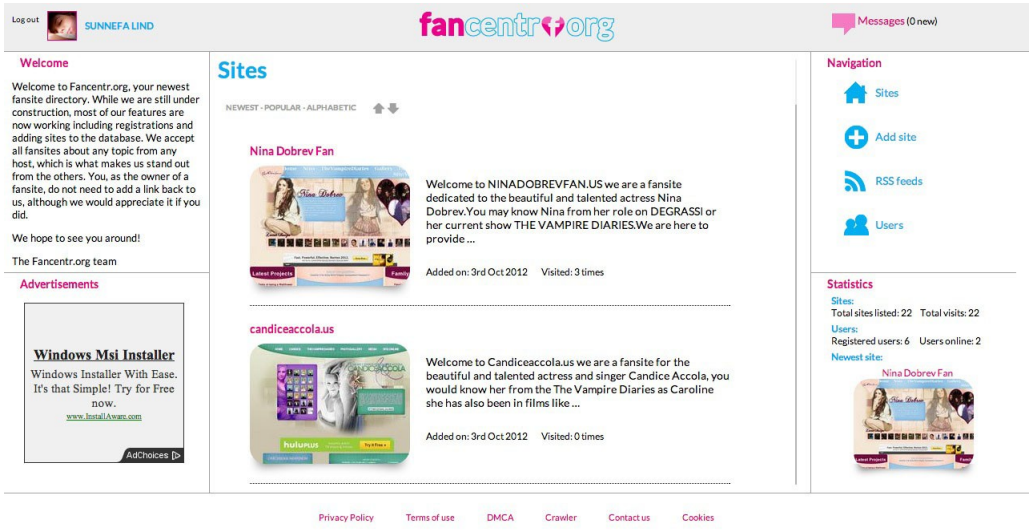
Mockup 3



Final implemented design

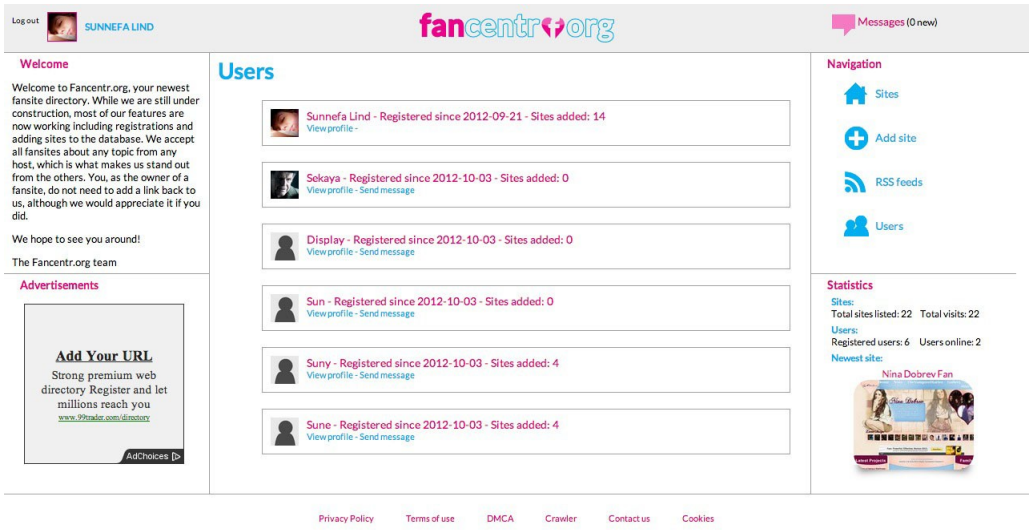
Out of the three mockups that I created I decided to implement the second one. The first and third ones have, in my opinion, too much of the pink colour and not enough of the blue. I also decided to have the header with a grey background, which I liked better than a white background as it sets the header apart from the rest of the site.

Home page / site list

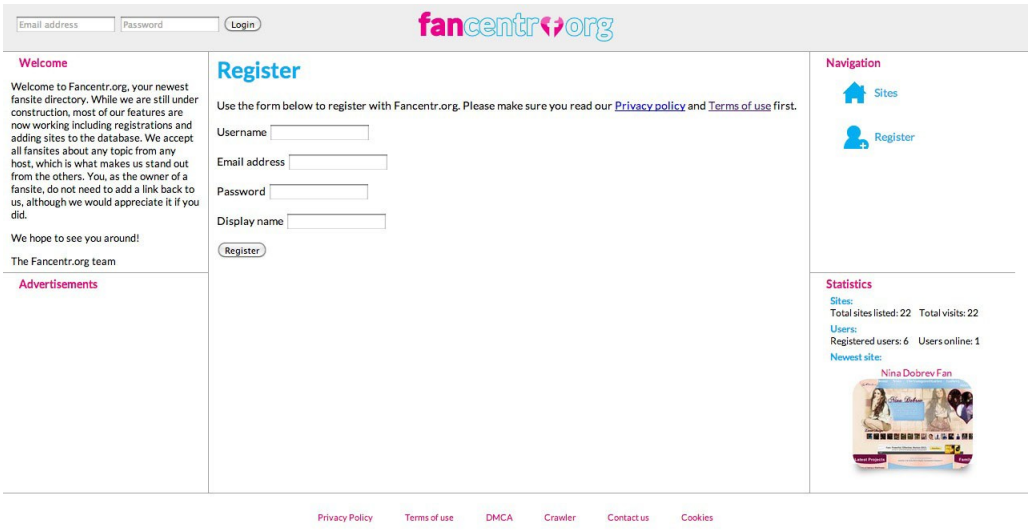


Add site

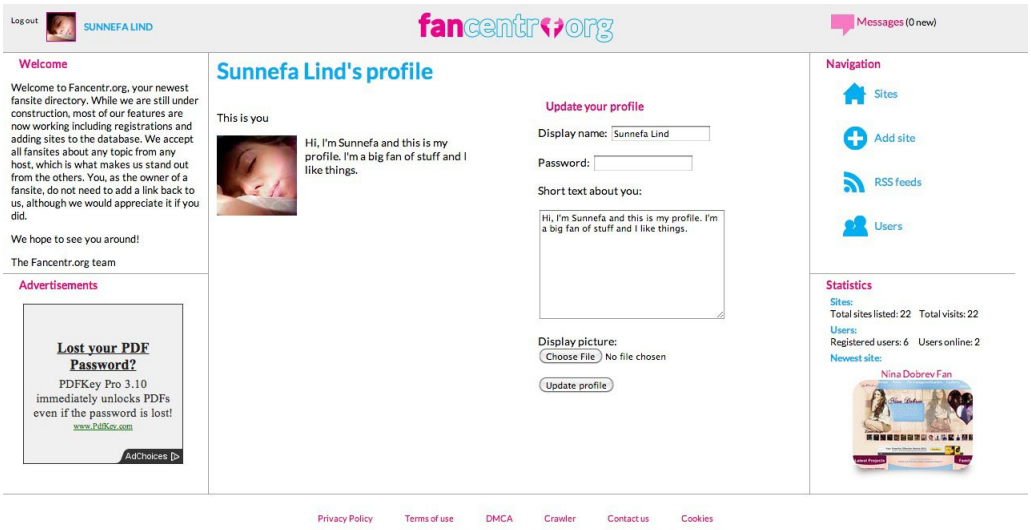
User list



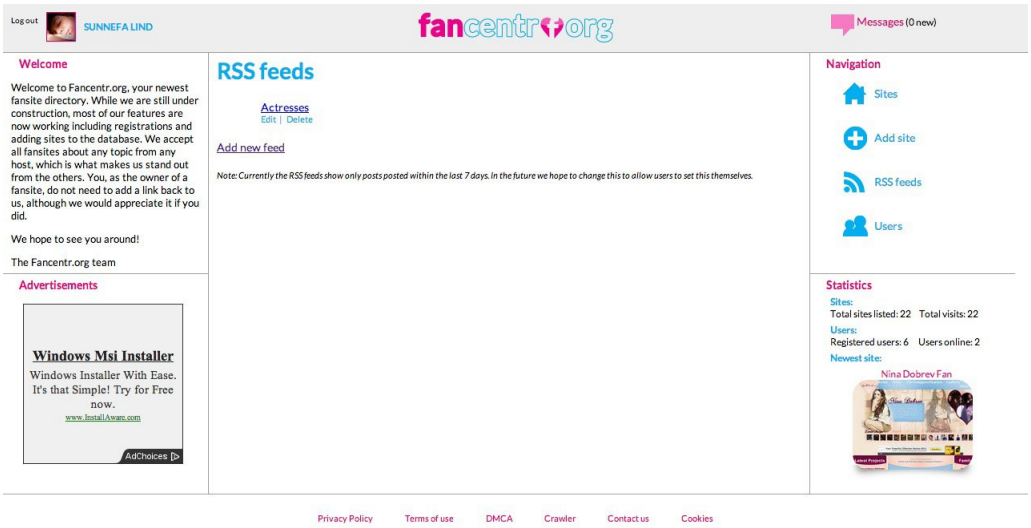
Register



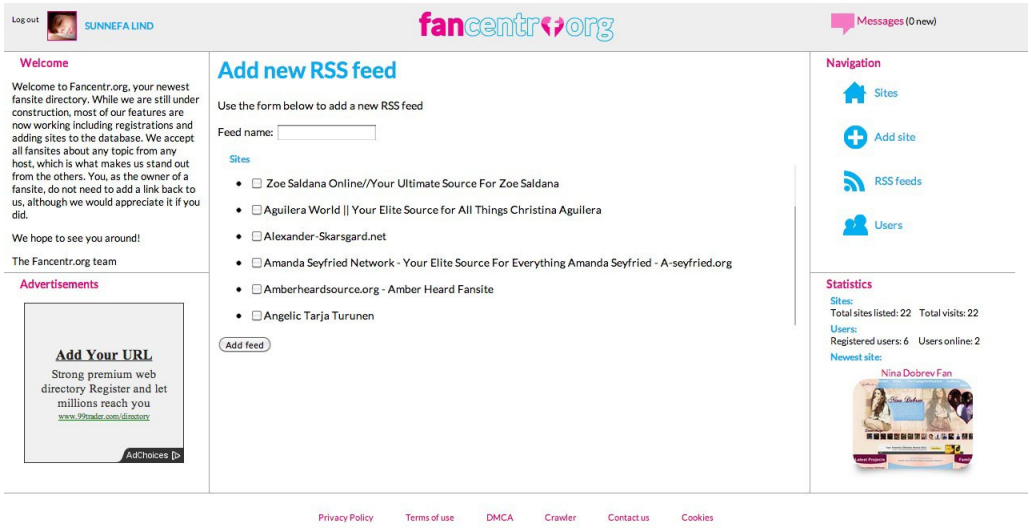
Profile



RSS feed list

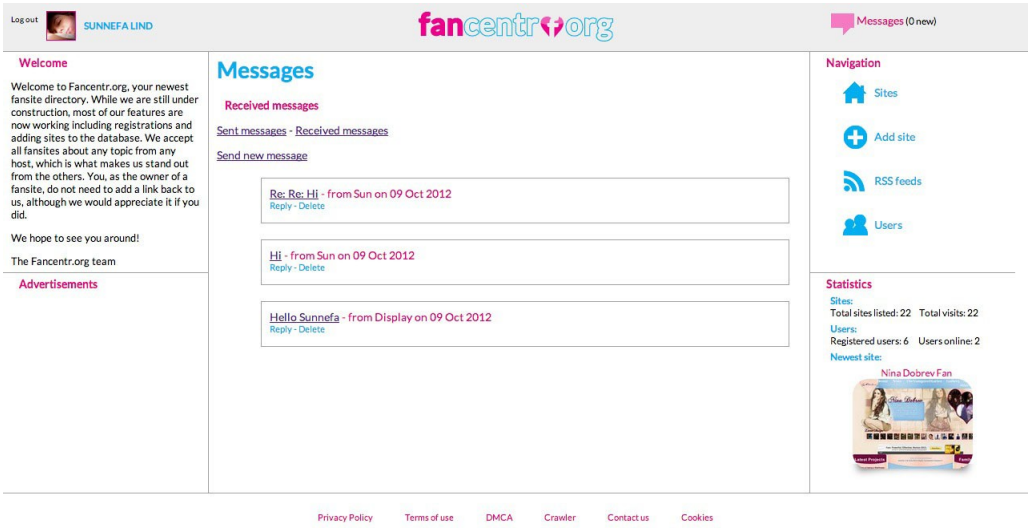


Create RSS feed

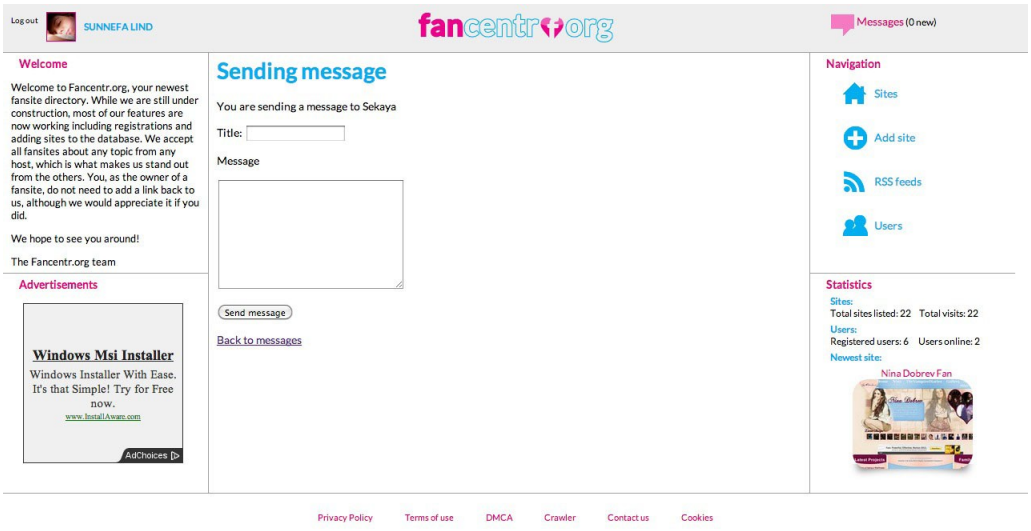


Read RSS feed

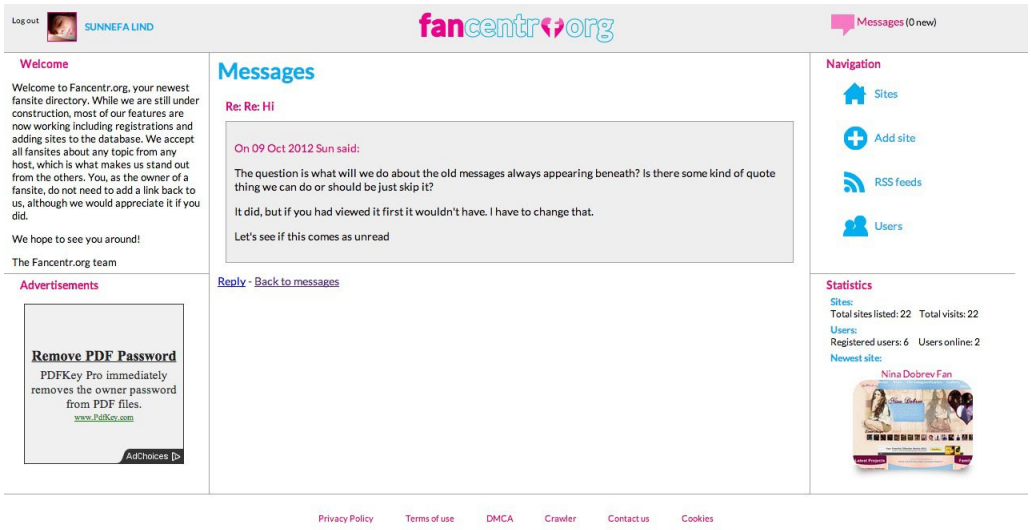
Messages List



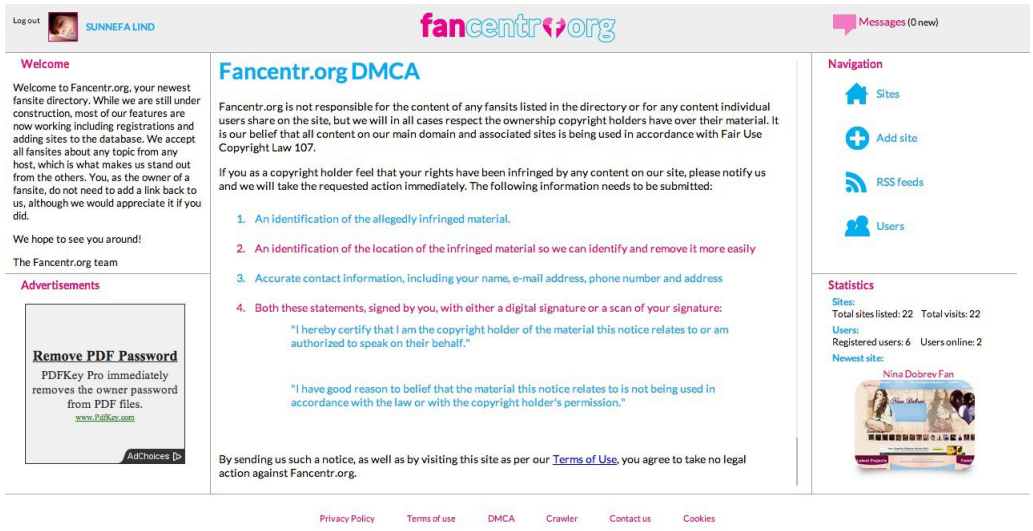
Send message



Read message



Text



Project programming stage

The project was programmed in PHP, coupled with HTML5, CSS3 and a little bit of JavaScript and jQuery. I followed the Model View Controller pattern as much as I was able to, keeping the data, logic and layout as separate as possible.

I developed one class for each table in my database, with methods for creating, retrieving, updating and deleting records. If the table called for it, the classes not only retrieved a single record from the database but all records in that table.

Some helper classes were needed that did not relate to the database. These included a class for uploading images, a class for validating data and a class for displaying screenshots of sites from an API.

I used the front controller pattern, meaning the project was programmed in such a way that all requests were sent through the main “index.php” file located in the “puclic_html” folder. Requests were divided into “POST” and “GET” requests which were handled by different controllers. Based on the exact arguments of the request these controllers then included the appropriate module controllers.

Each module had one controller for “GET” requests and another for “POST” requests. In some cases, however, these two controllers included the same file for a specific part of a module, such as the “add sites” part of the sites module, which handles both showing a form and processing the same form. This was partially because I had not decided beforehand how modules would handle “POST” requests.

In order to make the project more search engine friendly, I created a “.htaccess” file to rewrite the URLs of the project. Using the Apache “mod_rewrite” module I was able to write “rewrite rules” for the four main modules (sites, users, messages and RSS) and create URLs that looked like: “http://www.fancentr.org/users/profile/1” instead of looking like: “http://www.fancentr.org/index.php?page=users&part=profile&userid=1”.

Highlights and challenges

During the programming I did run into some problems. This meant I had to stop coding and do some research in order to solve my problem. Most of the time I was able to solve them in a few minutes, but there were times when it took longer than that. The most notable examples are highlighted below.

Crawler

The site crawler was the hardest thing to program. It both required a lot of research and took a lot of trial and error. Initially I used PHP’s ‘file_get_con-

tents" function to download the front page HTML of a site which I then sent to another function to analyse it. This was not a good idea because a lot of hosts disable this feature for security reasons. I had heard about cURL and experimented a little bit with it and I knew it could be used to communicate with remote servers. What I did not know was how exactly to use it.

cURL can be quite simple but also complicated. To begin with it needs to be initiated and told what URL to communicate with. Then it is possible to set some options such as what user agent to tell the server it represents, if it should make a "GET" or "POST" request, if it should follow redirects and it it should send any headers to the remote server. If cURL should save the data it gets back from the remote server, that must be specified as well. Then the request must be executed which, if specified, will return some data. In my case the data was the entire HTML structure of a site's home page.

Before receiving any data from cURL, I had to make sure that the URL was valid and that it existed. I created a function that uses cURL to check if the URL can be loaded. While it may not be perfect I have been able to write in a URL that I knew was not registered and get a negative response back.

After this check I checked if the URL was that of a site I already knew was not a fansite, such as Google, Facebook or the site itself (which would have passed the last test). This was done so that is a user attempted to put in such a URL I would not need to spend resources checking if it was a fansite, I could just deny the URL right away. The list is far from exhaustive and right up until the end I kept adding new sites to it.

If the URL was valid and was not on my exclusion list, I then needed to check if my program had permission to access the site. The program downloaded the site's "robots.txt" file, if it existed, and analysed it to check if it had been explicitly banned from accessing the site. If no such rule existed in the "robots.txt" file, or no such file existed at all, the program assumed it had permission. According to a tutorial I read all the major search engines work this way (SEOBook.com, 2011).

After receiving the data it could be sent to a function which would analyse the text and determine if the site was a fansite or not. If it turned out to be a fansite I then sent the same data to a function which extracted the site's meta description, title and RSS feed URL from the HTML source.

One of the challenges was with getting the description of the site. If the site had a meta description tag, it was no problem but for some reason that I cannot explain, a lot of fansites seem not to include it. It might be because a lot of fansite webmasters are inexperienced and do not know about this or they were unsure of what to write. To try to get around this issue I looked for a text that started with "Welcome to" because I know that is something most, if not all,

traditional fansites have in common. This worked quite well except of course when the site didn't have such a message, or worse, made the message an image, which is extremely common as well. In such cases I extracted the text from the first “<p>” tag I could find, which proved successful in some cases with getting some comprehensible text that was a part of the latest update but sometimes resulted in no text at all or something that made no sense. In one case the text I got back was “Powered by Affiliationally” (which is a popular link exchange script) and it seemed to be the only “<p>” tag on the page.

Because I anticipated cases such as this to be the exception rather than the rule, I stuck to what I had done and concluded it would be easier to change the description manually for the 1% of sites where this would happen.

The keyword analyses was another challenge because not all fansites have the word “fansite” written anywhere on it. Instead I made a list of words that I knew were common such as “fansite”, “fans”, “#1 source”, etc. and a site would pass as a fansite if it included one of those words. Unfortunately that did not work very well because an empty page could be made with only “fansite” written on it and my program would think it was a fansite. In the end I did some research to find more keywords and concluded that there were some keywords that appeared multiple times and others that often only appeared once. At first I attempted to make it so that at least two of the keywords from the word list had to appear twice and at least two of the others had to appear once, but this only worked for about two out of ten sites tested. In the end I combined the lists and made it so that three or more of these words had to appear or the site would not be considered a fansite.

This is of course not perfect, but worked in the majority of cases where the site was in English. Only accepting sites in English is very limiting to the number of sites can be added, but it would have taken too long to research what the keywords were in other languages such as German, Chinese, Spanish, Italian or Russian.

RSS feed reader

The RSS feed reader was another feature that also took a long time. I had to use cURL again, although because I had already made the crawler described above it was less of a challenge. The real challenge lay in working with PHP’s native SimpleXML object and navigating the RSS feed structure. I had initially planned to save the posts into my own database and periodically empty the table or remove any posts that were older than a certain age, but in the end I came to the conclusion that this might put too much of a strain on my database engine so I decided to load the posts from the RSS feeds every time. While this

approach takes a longer time, I believe it is better both because it is simpler and less resource consuming.

Unlike the site crawler where I put all the crawling functions into a separate class,when I was making the RSS feeds I did everything in the same class that I used to get the list of feeds from the database. A similar approach might have been taken as when making the crawler, but the method used proved a lot simpler to implement.

One of the things I was unable to figure out was how to indicate which feed had new posts since the last time the user viewed the list of feeds. I was unable to find a solution that did not require a long loading time. AJAX might have been up to the task, however, that would have required more refactoring of code than I had time for.

The RSS feeds are the only thing that I developed “CRUD” functions for. These functions allow users to add new feeds, edit them and delete them.

Messages

One of the last things I implemented was a private messaging system whereby users can keep in touch with each other. I had initially planned to make this into a sort of chatting feature where chat logs would be sent to users as messages (based on Facebook’s implementation). However, I had not anticipated how difficult and time consuming that would be. I ultimately decided to only do a simple messaging system. Users can send and receive messages and a link in the header notifies them if they have any new messages.

One of the challenges with this was how to make it so that users could reply to another message while keeping the same title and recipient. At first I tried saving this information in the “$_SESSION” superglobal, which works as long as the user does not refresh the page.

Because that is a poor usability decision, I made it so that the id of the previous message is sent in the “GET” request along with the id of the recipient. This meant I could load that message from the database and display it in the form. This was based on the messaging system at Red Dragon Inn.

Most of the private messaging systems I looked at allow the user to send messages to multiple recipients. Despite my attempts I failed to implement that feature. One way I could have done it is to add the ids of the users to the “GET” request, but this would have required an extremely complicated rewrite rule and would have resulted in a very long query string. Another way would have been to allow the user to write in the usernames of the recipients into a field, then loop through them to find the user ids. That method failed because each user has a display name that could possibly be the same as another user’s display name. The other users only ever see display names, not usernames, so I would

have had to change the user module drastically to make it so that no two users could have the same display name.

It would have been possible to allow the sender to select their recipients off a list, but this would have required I show all the users next to the message form every time, which I think would have been distracting to the user.

Fansite screenshots

Another thing which took a long time was the screenshots of the sites listed in the directory next to their description. This could have been skipped but doing so would have made the directory look less like a directory. At first I researched how I could do it on my own, but unfortunately that would have required the use of a Windows server and more resources than were available to me at the time (PHP Documentation Group, 2012). So instead I turned to online services that offer such screenshots for free. I found several, some of which were still in development and therefore did not work very well and others that claimed they were fully developed but still did not work very well. In the end, after spending time trying to get two others to work, I settled for Shrink the Web, which had been my original choice but I had looked for others that offered their premium features for free.

Shrink the Web has an API and when I registered I received an API key so I could start using their services immediately. The simplest way to get the screenshots was to make my images have a “src” attribute that pointed to their servers and format the URL to include the size of the image, my API key and the URL of the site I wanted a screenshot of. This, however, would have meant I would be making a lot of external requests and would not provide any kind of fallback in case their servers were down. Instead I wanted to get the image from Shrink the Web, save it on my own server and then serve that image to the user. The image would be valid for 3 days and the request to Shrink the Web would only be made if the image requested did not exist on my server or had expired.

At first I did this with PHP’s “file_get_contents” function, which was not a good idea because the “url_fopen” setting, which is required for this method to work, is disabled on a lot of hosts for security reasons and is less secure than cURL. The reason I did it this way was that I was looking at code provided by Shrink the Web which I had a hard time making sense of for lack of comments and what seemed to be intentional obscurity. In the end I found out that it was possible to make a request and get an XML object back which could tell me if the image I was requesting existed on Shrink the Web servers or not.

With the other method, if the image did not exist I would get back an image saying the thumbnail was queued but because of my caching I would have to

wait three days or manually change the cache time before I got the real image. With the XML way I could check if the image existed and only download it if it did. This way I could put in my own “coming soon” image which would then be replaced the next time the page was refreshed, if the thumbnail was ready at Shrink the Web. The only problem with Shrink the Web is that they have a waiting time for thumbnails and on a free account it is even longer. Also if the thumbnail did not exist before I requested it for the first time, it would affect how many more I could request and every image I loaded from their servers affected the amount of bandwidth they granted me. By saving the images on my own server I might be making it so that updates to a site’s layout take longer to show up on my site, but my bandwidth with Shrink the Web is not affected as much so I can serve more screenshots.

Possibly in the future I would like to develop my own way of taking screenshots of sites, but this might require programming knowledge that is far beyond the scope of this degree.

Conclusion

This essay has detailed what research I carried out and my findings, and how I used that to create the fansite directory platform. I have detailed exactly what features it has and how I went about implementing them, why I chose those features and how I planned them. I have also shown how I designed the directory.

While this project did not become everything I wanted it to be when I was making it, I believe it has a lot of potential. There is much room for expansion and it is programmed in such a way that adding new features would not require major alterations of the features it already has.

At the same time I believe I have done the very best job I possibly could with this project and have illustrated in detail why I made the decisions I made during the development and design stages.

My efforts resulted in a fansite directory that includes social networking features. It allows users to discover new fansites, connect with other users through messages and share their favourite fansites with them. Users can also create lists of sites they visit regularly and check if they have any new posts by use of an RSS feed reader. The directory is a good place to find new fansites and share the ones that you enjoy.

List of references

| Celeb Listed | Directory of quality fansites...Best guide for all celebrity fansites on the net with quality content. (no date) Available at: http://www.celeblisted.-com/submit.php (Accessed 27 July 2012).

Banda, S. (2012) Introduction to Object Oriented Programming [Lecture for BA/BSc Web Design and Development Year 2]. 18 January.

Facebook (2012) *Facebook*. Available at: http://www.facebook.com (Accessed 29 September 2012).

Fan Sites Network (2012) *Fan Sites Network Hosted Sites: Main*. Available at: http://fan-sites.org/hosted-sites/ (Accessed 17 October 2012).

Goodreads Inc (2012) *Share Book Recommendations With Your Friends, Join Book Clubs, Answer Trivia*. Available at: http://www.goodreads.com (Accessed 27 September 2012).

Peacock, M. (2010) *PHP 5 Social Networking*. *Amazon.co.uk* [e-book reader]. Available at: http://www.amazon.co.uk/PHP-5-Social-Networking-ebook/dp/B0057G9RQK/ref=tmm_kin_title_0?ie=UTF8&m=A3TVV12T0I6NSM&qid=1328577244&sr=1-10 (Accessed 19 October 2012).

PHP Documentation Group (2012) *PHP: curl_init - Manual*. Available at: http://www.php.net/manual/en/function.curl-init.php (Accessed 20 October 2012).

PHP Documentation Group (2012) *PHP: imagegrabscreen - Manual*. Available at: http://php.net/manual/en/function.imagegrabscreen.php (Accessed 20 October 2012).

PHP Documentation Group (2012) *PHP: PHP Manual – Manual.* Available at: http://www.php.net/manual/en/ (Accessed 20 October 2012).

Pokemon Fansite Directory || The Pokemon Desert Shadow (no date) Available at: http://pokeds.com/directory/ (Accessed 15 May 2012).

SEOBook.com (2011) *Robots.txt Tutorial*. Available at: http://tools.seobook.com/robots-txt/ (Accessed 15 September 2012).

Starszz.com (2012) Starszz.com Hosted Sites Directory. Available at: http://sites-directory.starszz.com/ (Accessed 15 October 2012).

The jQuery Project (2010) *jQuery.ajax() - jQuery API*. Available at: http://api.jquery.com/jQuery.ajax/ (Accessed 27 August 2012).

The jQuery Project (2010) *jQuery Wiki*. Available at: http://docs.jquery.com (Accessed 29 September 2012).

Twitter (2012) *Twitter.* Available at: http://twitter.com (Accessed 27 September 2012).

Zandstra, M. (2010) PHP Objects, Patterns and Practice (Expert's Voice in Open Source). Amazon.co.uk [e-book reader]. Available at: http://www.amazon.co.uk/Objects-Patterns-Practice-Experts-ebook/dp/B004VJ2YZU/ref=sr_1_1?s=digital-text&ie=UTF8&qid=1329142215&sr=1-1 (Accessed 19 October 2012).

Zuckerberg, M. (2012) ‘This morning there...’, Facebook, 4 October. Available at: http://www.facebook.com/zuck/posts/10100518568346671 (Accessed 21 October 2012).

Bibliography

| Celeb Listed | Directory of quality fansites...Best guide for all celebrity fansites on the net with quality content. (no date) Available at: http://www.celeblisted.-com/submit.php (Accessed 27 July 2012).

Banda, S. (2012) Introduction to Object Oriented Programming [Lecture for BA/BSc Web Design and Development Year 2]. 18 January.

Em Calculator (no date) Available at: http://riddle.pl/emcalc/ (Accessed 31 August 2012).

Facebook (2012) *Facebook*. Available at: http://www.facebook.com (Accessed 29 September 2012).

Fan Sites Network (2012) *Fan Sites Network Hosted Sites: Main*. Available at: http://fan-sites.org/hosted-sites/ (Accessed 17 October 2012).

Goodreads Inc (2012) *Share Book Recommendations With Your Friends, Join Book Clubs, Answer Trivia*. Available at: http://www.goodreads.com (Accessed 27 September 2012).

Harris, T. (no date) *Create UML diagrams online in seconds, no special tools needed*. Available at: http://yuml.me/ (Accessed 12 October 2012).

i-ssential GmbH (2008) *thumbalizr - thumb your webpages*. Available at: http://www.thumbalizr.com/ (Accessed 14 September 2012).

Lazaris, L. (2011) ‘Learning To Use The :before And :after Pseudo-elements In CSS’, *Smashing Coding*, 13 July. Available at: http://coding.smashing-magazine.com/2011/07/13/learning-to-use-the-before-and-after-pseudo-elements-in-css/ (Accessed 5 September 2012).

Lennartz, S. (2009) ‘The Mystery Of CSS Sprites: Techniques, Tools And Tutorials’, *Smashing Coding*, 27 April. Available at: http://coding.smashingmagazine.-com/2009/04/27/the-mystery-of-css-sprites-techniques-tools-and-tutorials/ (Accessed 1 September 2012).

Malihutsakis, M. (2012) 'jQuery custom content scroller', *malihu | web design*, 27 July. Available at: http://manos.malihu.gr/jquery-custom-content-scroller (Accessed 31 August 2012).

Peacock, M. (2010) *PHP 5 Social Networking*. *Amazon.co.uk* [e-book reader]. Available at: http://www.amazon.co.uk/PHP-5-Social-Networking-ebook/dp/B0057G9RQK/ref=tmm_kin_title_0?ie=UTF8&m=A3TVV12T0I6NSM&qid=1328577244&sr=1-10 (Accessed 19 October 2012).

Pears, R. and Shields, G. (2010) *Cite Them Right*. 8th edn. New York: Palgrave Macmillan.

PHP Documentation Group (2012) *PHP: curl_init - Manual*. Available at: http://www.php.net/manual/en/function.curl-init.php (Accessed 20 October 2012).

PHP Documentation Group (2012) *PHP: imagegrabscreen - Manual*. Available at: http://php.net/manual/en/function.imagegrabscreen.php (Accessed 20 October 2012).

PHP Documentation Group (2012) *PHP: PHP Manual – Manual.* Available at: http://www.php.net/manual/en/ (Accessed 20 October 2012).

Pokemon Fansite Directory || The Pokemon Desert Shadow (no date) Available at: http://pokeds.com/directory/ (Accessed 15 May 2012).

SEOBook.com (2011) *Robots.txt Tutorial*. Available at: http://tools.seobook.com/robots-txt/ (Accessed 15 September 2012).

ShrinkTheWeb (no date) *Website Screenshot Thumbnail Service | Shrink-TheWeb*. Available at: http://www.shrinktheweb.com/ (Accessed 26 September 2012).

Sinnott, G. (2011) *Spritebox - Create CSS from Sprite Images*. Available at: http://www.spritebox.net/ (Accessed 1 September 2012).

Starszz.com (2012) Starszz.com Hosted Sites Directory. Available at: http://sites-directory.starszz.com/ (Accessed 15 October 2012).

Taylor, M. (2008) ‘Equal Height Columns with Cross-browser CSS’, *Art and Design Blog by Matthew James Taylor*, 18 October. Available at: http://matthew-jamestaylor.com/blog/equal-height-columns-cross-browser-css-no-hacks (Accessed 25 August 2012).

The jQuery Project (2010) *jQuery Wiki*. Available at: http://docs.jquery.com (Accessed 29 September 2012).

The jQuery Project (2010) *jQuery.ajax() - jQuery API*. Available at: http://api.jquery.com/jQuery.ajax/ (Accessed 27 August 2012).

The Red Dragon Inn - resources for fantasy and D&D fans! (no date) Available at: http://www.rdinn.com/ (Accessed 10 September 2012).

Thumbshots.com (no date) *Free preview thumbnails on your site. Web page screenshot image*. Available at: http://www.thumbshots.com/ (Accessed 15 September 2012).

Twitter (2012) *Twitter.* Available at: http://twitter.com (Accessed 27 September 2012).

www.tarja-turunen.com • Index page (no date) Available at: http://www.tarja-turunen.com/forum/ (Accessed 11 September 2012).

Zandstra, M. (2010) PHP Objects, Patterns and Practice (Expert's Voice in Open Source). Amazon.co.uk [e-book reader]. Available at: http://www.amazon.co.uk/Objects-Patterns-Practice-Experts-ebook/dp/B004VJ2YZU/ref=sr_1_1?s=digital-text&ie=UTF8&qid=1329142215&sr=1-1 (Accessed 19 October 2012).

Zuckerberg, M. (2012) ‘This morning there...’, Facebook, 4 October. Available at: http://www.facebook.com/zuck/posts/10100518568346671 (Accessed 21 October 2012).

Appendix I – Fansite directory web application

http://beta.fancentr.org