Adobe Flex / Zend for Content Management

Master-Project Thesis

Document code	: Thesis_ska310_v2.0.pdf
Version	: v2.0
Created by	: Stefan van de Kaa
Date:	: June 17, 2010
Supervisor	: Prof. Dr. A. Eliëns
Second reader	: Dr. R. Premraj
Master	: Computer Science
Specialization	: Multimedia

Abstract

Since the beginning of times information and content was shared, there was a need to manage this information. Nowadays there are computers and systems which are able to manage the content inserted by the users. These systems are Content Management Systems (CMS's). This thesis describes how and why Adobe Flex and The Zend Framework can improve this management.

Adobe Flex is a programming language which displays information. This information can be retrieved with another language. For data retrieval The Zend Framework can be used. Together Flex and Zend can create rich internet applications. A rich internet application is an application which can be ran from within any browser. The look and feel of the application is not depending on the browser itself and will be rendered exactly identical. For a general idea about The Zend Framework, Adobe Flex and Adobe Flash this document contains information about the background and history of these technologies.

Before starting with the implementation of a custom CMS research has been done. Three different CMS's have been reviewed and compared. Their positive and negative points have been described. The CMS's reviewed are Joomla!, Drupal and WordPress.

After reviewing three different CMS's we defined requirements for a custom CMS. Implementing the CMS resulted into several issues which needed to be resolved in order to successfully finish the project. At the moment of writing there is, to my knowledge, no CMS similar to our CMS.

To addition, there is a tutorial which contains the knowledge which is required to implement a basic CMS with Adobe Flex and The Zend Framework. This includes the creation, installation and configuration of a Zend application.

Preface

Prior to this project there were communication issues which resulted into a failure for the first version of this document. Within the first version I created a complex and working Content Management System but neglected to do proper research about other systems and about the requirements of the CMS. I would like to thank Ronald Siebes and Anton Eliëns for their criticism.

This second version includes reviewing other CMS's and includes a tutorial about how to implement a basic CMS without knowledge. Writing this second document took a lot of extra time, but it was worth the effort since I ran across many interesting details concerning content management systems and their implementations.

Table of Contents

1 Introduction

The World Wide Web is a growing medium which is gaining popularity each day. Every single day new applications are being developed and new ideas evolve. At the current time users of systems are relying more and more on their software systems and therefore want to be able to access the application from various locations at different times. These applications are more than just a general website, they are complex and share delegate information with multiple users. One of the possibilities for creating such applications is Adobe Flex. Adobe calls these applications Rich Internet Applications (RIA) (1).

The downside of working with Adobe Flex is that Flex is a tool for displaying data where it isn't able to retrieve the information from for example a database. For retrieving the data The Zend Framework comes in place. The Zend Framework is an open source PHP Framework which allows the developer to set up a website within a reasonable timeframe. Zend can do this because there a large amount of modules included inside the framework which are ready for use.

Knowing this information rises the following question: "why The Zend Framework?". I have had experience with Adobe Flex and also with The Zend Framework and became curious about the possibilities for creating a CMS with it.

The research issues described in this thesis are:

- How can a Flex Application improve the usability including the "look and feel" for users of a Content Management System?
- Why create a custom CMS when there are open source systems to choose from?
- What are the most important properties of a CMS which should be available within any decent CMS?

Below is an overview about the chapters inside this thesis and their contents.

- Chapter two:
  This chapter describes the background of Adobe Flex and The Zend Framework. Further this chapter will review the reasons for developing a custom CMS.
- Chapter three:
  Within this chapter three currently existing CMS's are reviewed. Their positive and negative points will be addressed and a comparison will be made. This will give a good impression about the features which the custom CMS should include.
- Chapter four:
  During this project a custom CMS is developed. This chapter describes the features and requirements which are required for the CMS.
- Chapter five:
  This chapter describes the problems and issues which occurred during the implementation of the custom CMS. Further the choices made when a problem occurred are described with their argumentation.

- Chapter six:
  The evaluation of the overall project takes place in chapter six. This includes the custom CMS, the tutorial and my personal opinions about the Zend Framework and Adobe Flex.
- Chapter seven:
  The conclusion of this project is described in the final chapter, chapter seven.

Terms and Definitions

Below are the terms and definitions stated which are used inside this document.

Content Management System (CMS):

A CMS is a system to edit or manage the content of a system. This can be a website or for example a word document. According to Wikipedia the CMS's described in this document are Web CMS's. (2).

Client and Server

The term client in this document is used for the computer of the end-user. This means the computer of the person using the CMS system. The term server is used for the computer where the data of the website is stored and where the website is hosted on.

User, End-user and Developer

When using a CMS there are several users in play. The first type of users are the normal Users. These are the persons which are using the publishing website. The End-user is the person which is editing the website and changing/managing the data. The Developer is the person which has created the CMS itself.

Frontend vs Backend

The frontend of a system as mentioned in this document is the website which is displaying the content where the backend is the system which can change the information. This is the actual CMS. Inside the tutorial the frontend is referred to as the publishing website to make the tutorial as straightforward as possible.

2 Background

This chapter describes the history of this thesis in combination with the history of the Zend Framework, the history of Adobe Flex and the Adobe Flash Player.

2.1 Why a custom CMS

When you have a website which needs to be managed there are multiple solutions which might be interesting. For instance it is an option to edit the HTML codes and upload the source to the web-server once the changes have been made. Another solution is the use of a Content Management System. Such a system can be used to manage the content on your website. These systems come in different types with different possibilities. For instance you can buy a CMS to use or select an open source framework which suits your needs.

Since I want to have the control over my website and don't want to be reliable on other people I chose to develop my custom CMS. The advantage of this is that besides the control over the CMS you know everything about the system which makes it easily to add and remove functionality. Also security problems can be removed easily if there are any.

Finally, maybe the most important reason for me to develop a CMS myself is that I haven't found any CMS which is developed with Adobe Flex. In my opinion Flex has enough features and options to create a CMS which is very user friendly and user-friendliness should be the main concern for a solid CMS.

2.2 The Zend Framework

When systems get larger and in addition more complex programmers might create code which is not as efficient as it should be. There are several solutions to prevent this. One of the best is programming in an Object Oriented manner. Further you don't want to invent the wheel again and therefore you can use frameworks.

*A **framework** is a basic conceptual structure used to solve or address complex issues. This very broad definition has allowed the term to be used as a buzzword, especially in a software context.* (3).

The Zend Framework is a PHP framework which has a lot of features included to make it easier for the programmer to communicate with other software languages and protocols. These available communication layers are structured in such a way that it works very generic and similar.

Besides these layers the Zend Framework also includes a lot of modules which the programmer might want to use with his application. For example the GData API (from Google), Flickr, Yahoo and Twitter. So when setting up a larger application like an CMS it is good to use a Framework which improves the writing of cleaner and better source code and also because of the modules and includes which are already included inside the framework.

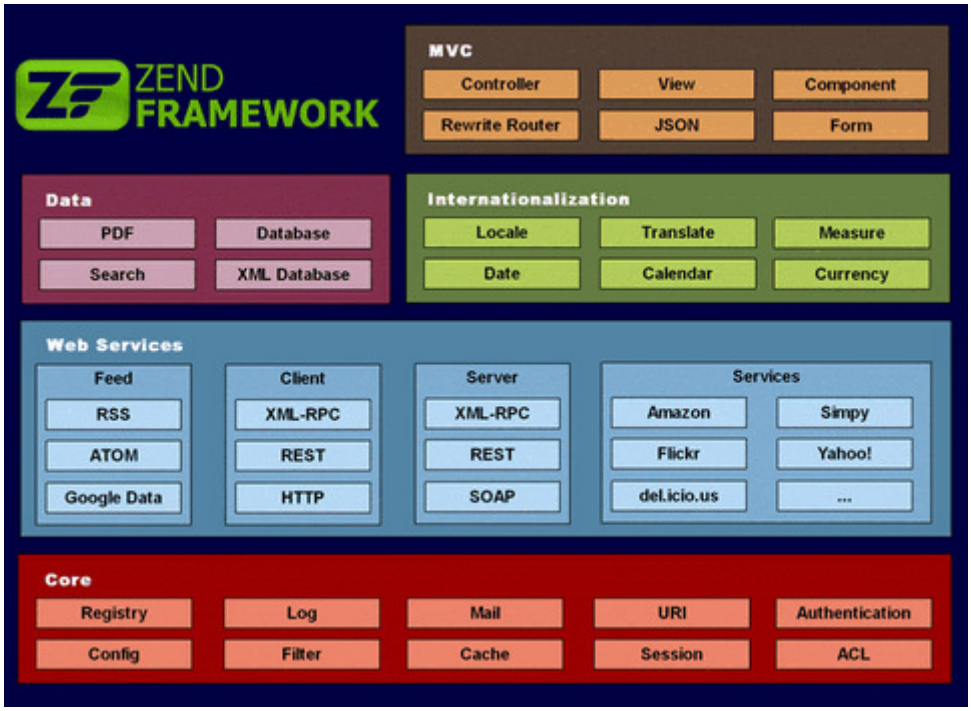


Figure 2.2.1: Zend Framework Features, http://tweakers.net/meuktracker/21525/zend-framework-193-pl1.html

The image above described the features and modules of the Zend Framework. More detailed information about the modules and choice for the Zend Framework can be found in the Literature Study which is attached to this document.

2.3 Adobe Flex

Adobe Flex is an open source programming language to create applications which run inside the Flash Player. The Flash Player is another product of Adobe. The Flash Player is software which can be used in for example browsers to display Flash applications. At the moment of writing 98.9% of all the computers around the world have the Flash Player installed (4).

Flex applications can be developed with any text editor. For example WordPad or Notepad. Adobe has however also developed a tool to rapidly develop Flex applications. Previously this tool was called the Flex Builder. This year (2009) the naming of the Flex products has changed and the builder is renamed to Flash Builder.

Applications created with Adobe Flex have a look and feel of a desktop application and are therefore familiar to the most users. This improves the user friendliness. The programmer also benefits from Flex since there are a lot of components included into the SDK which improves the speed of application development.

There is a good reason for Adobe and Macromedia to start developing Flex. Adobe Flash, the other tool to create Flash applications was developed for designers. Developers therefore had problems trying to understand how Adobe Flash works.

Where Adobe Flash uses a timeline to create the structure of the application, Adobe Flex uses another programming language called MXML. MXML is a specific XML markup. MXML in combination with ActionScript, which is also used with Adobe Flash, can be compiled into a Flash application.

The first company which started with Flex was Macromedia. Adobe continued with developing Adobe Flex after the takeover of Macromedia on April 18th, 2005 (5). Adobe however changed the licensing policy and distributed the SDK as open source.

At the moment version 3.4 is the latest release. Adobe is planning to release version 4 (Gumbo) in early 2010. Below detailed information about the version history (6).

- Flex 1.0 – March 2004
- Flex 1.5 – October 2004
- Flex 2.0 – June 28, 2006
- Flex 2.0.1 – January 5, 2007
- Flex 3.0 – February 25, 2008
- Flex 3.1 – August 15, 2008
- Flex 3.2 – November 17, 2008
- Flex 3.3 – March 4, 2009
- Flex 3.4 - August 18, 2009
- Flex 4 – 2010

Figure 2.3.1: Adobe Flex Logo, http://en.wikipedia.org/wiki/File:Flexicon.png

3 Well known Content Management Systems

There are several different Content Management Systems available online. When searching on the internet for free CMS's there are a few systems that you will notice in the first hits of the results. These systems are: Joomla!, Drupal and WordPress.

Since I didn't want to compare just the first hits of Google I searched for the best free CMS's. This resulted in a list of the top forty of best free CMS systems (7). Within this list Joomla!, Drupal and WordPress are in the top three. Since this is only one comparison I also searched for other CMS comparison websites. At webdevnews.net (8) there was another comparison with a top ten of open source CMS's. In this comparison we also see Drupal, WordPress and Joomla! But only in another ranking.

Because these CMS's are very popular and well known I will have a review of these systems and compare them with each other. Doing this will give a good idea about the parts which are important within a decent CMS.

3.1 Comparison criteria

For comparing these three CMS's we need some criteria. In the next chapters I will go into the criteria which I will go into when reviewing the CMS's.

3.1.1 Installation

When a user wants to set up his website and wants to use a CMS he or she has to install the package onto his server. Since every package might have different requirements for installing the package we will have a look at the installation process of the CMS. Another thing to look at if the installation process is easy and user-friendly. The best solutions would be pushing a button and everything would be set up.

3.1.2 Usability

The second thing I will look at is the usability of the CMS. This means setting up your first website, changing the contents of the website and changing the types of content. For example a blog on the website or adding pages with the latest news. Finally an overall review of the usage of the framework and for which type of user and website this CMS would be perfect.

3.1.3 Search Engine Optimization

Everyone who wants his own website wants to make sure his or her website will be found when searching for in with search engines. There are a lot of options to make sure your website will be higher in the results. I will check what these CMS's have included to help you get higher in the search results.

3.1.4 Extendability

One of the most important features of a CMS would be if it is extendable. For example if we have a working system and we want to include a forum or a photo gallery. In this part I will check the possibilities of adding modules to the CMS and how easy or difficult this is. Since this has to do with the source code of the framework I will also go into the required knowledge when extending the CMS to suit the needs of the user.

3.1.5 Templating

When working with a CMS it's required to work with templates to use the design you want for you website. Since it might be very complex to create the website in the style you want I want to check out how difficult this is in reality for these CMS's.

3.2 Joomla!

For starting I want to give an introduction to what Joomla! is and where it came from. Joomla! is a CMS under the GNU/GPL License. Next to this it works as an application framework. This because the CMS can be used with a lot of plug-ins which are freely available.

Figure 3.2.1: Joomla! logo, http://cdn.joomla.org/images/logo.png

So how did Joomla! get where it is today? In march 2000 a company called Miro Construct Pty Ltd started developing a closed source project called Mambo (9). In 2002 this project became open source. In August 2005 a new project is started called Joomla!. This project started with the same codebase as the current Mambo project (10).

Joomla! has been rewarded with multiple awards. Joomla! won the Packt Publishing Open Source Content Management System Award in both 2006 and 2007 (10). Joomla! is also nominated for the best open source CMS winner of 2009 (11).

3.2.1 Installation

Before installing we need to know about the system requirements of the CMS. The requirements of the Joomla! CMS can be found on their website and are very clear (12). Since I will work with an installation of WAMP (13) all the required needs are met automatically.

We can get the latest version of the Joomla! CMS on the download page (14). I used version 1.5.14 for this document. This is a zip file which can be opened with WinZip, 7Zip of any other (de-)compression tool. The extracted folder needs to be placed inside the document root of the web server. With WAMP on windows the default location for this is c:\wamp\www. Since the CMS there needs to be a database which is set up. This can be done with PhpMyAdmin which is integrated with WAMP.

If we then browse to the website which would be located at: http://localhost/Joomla_1.5.14-Stable-Full_Package/ then we would automatically see an installation wizard which is visible in figure 3.2.1.1. In the second step the installation will check if the requirements for the installation are met which is a very nice feature.

The only real thing we need to do is fill in the settings for the correct database which will be validated during the installation. We can set up an FTP if we want. I decided to skip this feature. Another nice feature is to install data which makes it easier to start working with the CMS as a new user. Then the installation asks to delete the installation folder on the server. When this is done we have a fully working website which can be adjusted to suit the needs of the user.

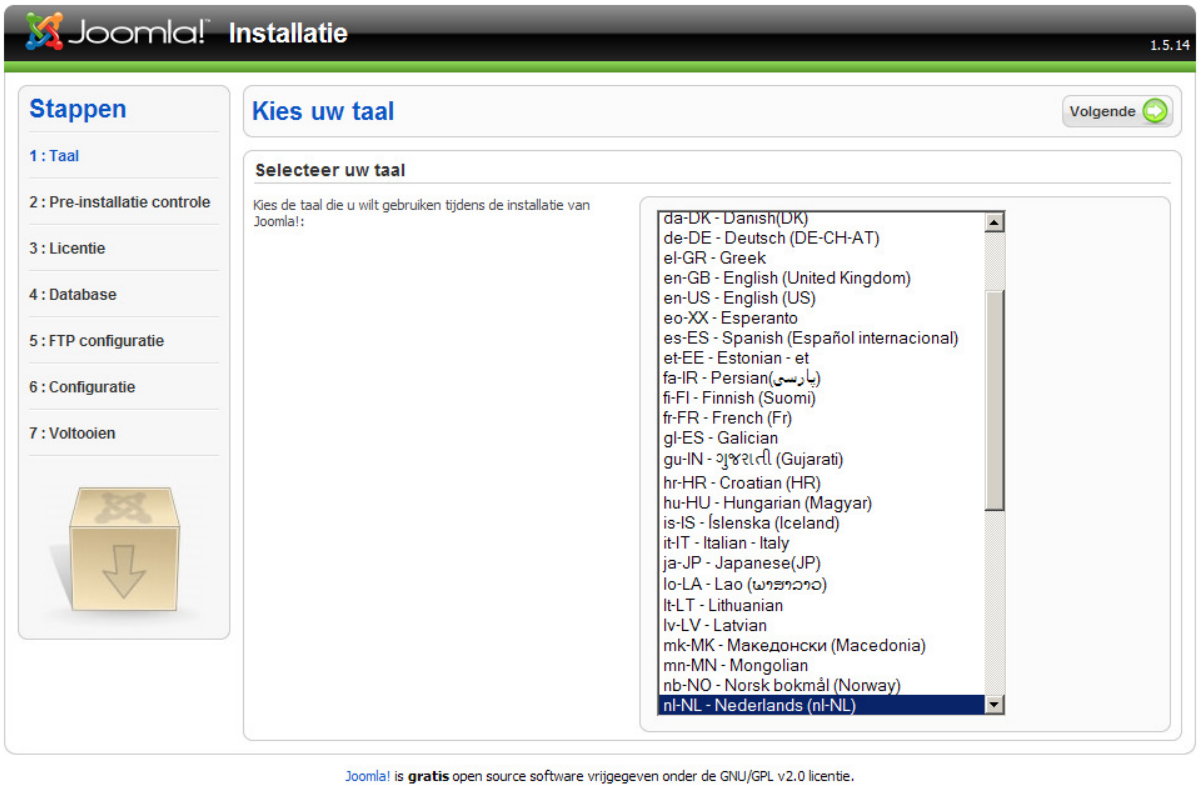


Figure 3.2.1.1: Joomla Installation.

3.2.2 Usability

To test if the Joomla! CMS is user-friendly I will have to test the functionality which is included inside the system. Therefore I will add a page to the homepage. The first thing which I want to do when I see the administrator panel is write a new article. This seems to me a new page, so this is what I will do.

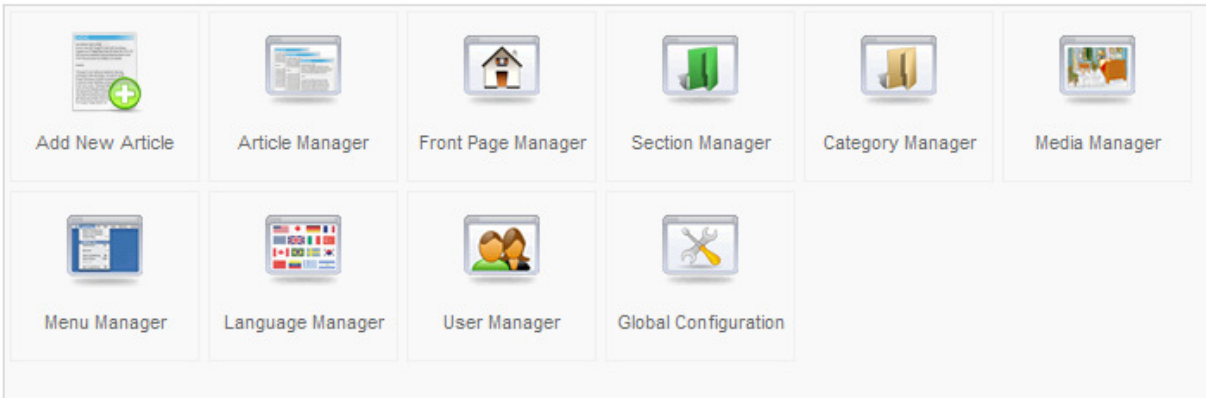


Figure 3.2.2.1: Joomla administrator panel

When adding the article we have to provide some information like if we want to display it on the homepage and if it should already be published. Also we can define which category we want this article be part of. Or if the article should be uncategorized.

So how does this article system works? When working with the CMS I find it unclear how it actually works. I know that we have sections, categories and articles. After searching online I found this image which clarified a lot for me.

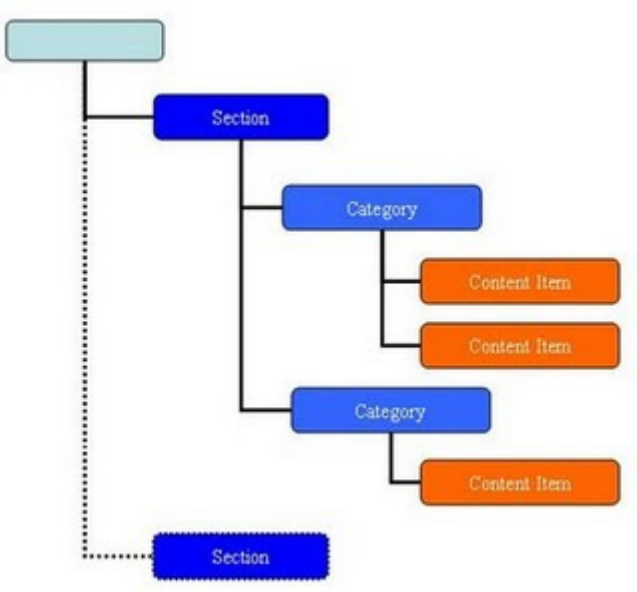


Figure 3.2.2.2: Joomla! structure, http://2.bp.blogspot.com/_1rl-4SPNYjM/SiE6eS5N0MI/AAAAAAAAAD0/-Hv9YyQh8hA/s320/26.jpg

Here we can see that if we have a complete website this contains numerous of sections. Each section can contain multiple categories. And each category in his place can contain multiple content items which can be articles. Also can a content item be for example a menu or a login form. Knowing this makes working with the CMS a lot easier.

Since there are a lot of extra features which you will not find out by just using the CMS it is good to read the quick start manual which can be found on the website.

http://help.joomla.org/ghop/feb2008/task048/joomla_15_quickstart.pdf

3.2.3 Search Engine Optimization

Since SEO is very important for the next generation of websites also CMS's have to be prepared for this. By default Joomla! has the option for working with SEO disabled. For example, there is an integrated solution which rewrites the urls to Search Engine Friendly Urls, but this is disabled by default. (15), (16)

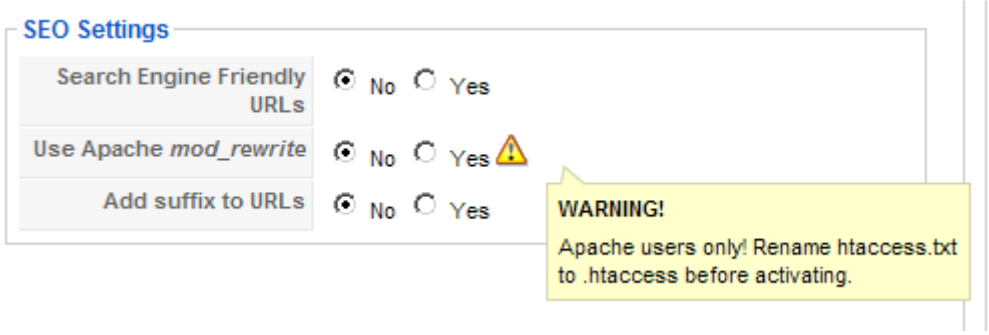


Figure 3.2.3.1: SEO Settings

The reason this feature is disabled is because Joomla! can also be used on another web-server than Apache and these other servers might require different rewrite rules which are not implemented in the release of the CMS. When enabling the SEO setting you do need to enable the mod_rewrite in Apaches configuration and rename the already provided htaccess file.

Further with all websites it is important to set the right keywords and title for each page. This makes the position inside search engines higher because they can be indexed in a better way. There is a patch available for Joomla! which gives you control to the metadata and title tags inside the page. This way you can make sure your page gets higher inside the search engines. (17)

3.2.4 Extendability

Expanding the Joomla! CMS isn't very difficult. There are various extensions available which can be included into the CMS. (18) You should however be warned that these extensions are mostly developed by normal programmers and therefore might contain bugs and don't have any warranty.

The good thing to do before using these extensions is to create a backup of your website and then install the extensions and then do some extensive testing. Even then there might be some security risks or bugs inside the code.

If you are really into the code of Joomla! you can create your own extensions and use these on your website. There are several guides on how to do this. (19) By creating your own extensions you can make sure that you have secure code and remove the bugs easily since the code is your own.

Figure 3.2.4.1: VirtueMart Joomla! extension,
http://booksr.netau.net/index.php/component/virtuemart/?page=shop.browse&category_id=2&vmcchk=1

Above is an example of a Joomla! extension which is included in a live environment. Including such a web shop inside the Joomla! CMS system is very easy.

3.2.5 Templating

When working with a website you want to give it your own look and feel. There are two default templates installed when you install Joomla! to your web server. But because your website should look unique you can download several other templates from the web freely (20). But if you choose to create your own template this is also a possibility. Doing this requires knowledge of HTML, PHP and CSS. There are detailed guides about how to create a template which is suitable for the Joomla! CMS (21).

3.3 Drupal

How did Drupal come to where it is today? Dries Buytaert was working on a bulletin board which evolved into the CMS which it is today. In 2001 the project became open source and at this moment the community is also developing with the CMS (22).

Drupal have been rewarded with the 2007 Overall Open Source CMS Award. Also did the system have been finished in second place for the *Best PHP Open Source CMS* nomination and the *Best Open Source Social Networking CMS* nomination (22).

Figure 3.3.1: Drupal logo, http://culturekitchen.com/files/drupal.jpg

3.3.1 Installation

Before installing the Drupal CMS we need to know about the requirements of the Drupal system. The requirements can be found on their website after searching for a while (23). Working with the WAMP web server system all the requirements are met automatically. We can however enable some features as user friendly urls when we enable the mod_rewrite module in apache.

Downloading Drupal can be done via their download page which isn't that easy to find (24). But once located the page we can easily download the archive needed to set up the CMS. For this document I used version 6.14 which is at moment of writing the latest official release. When we extract the archive with an (de)compression tool and place this in the folder of the web server we can visit the page and follow the installation instructions. Also we need to set up a database via PhpMyAdmin.

When we navigate to the url of the CMS (http://localhost/drupal-6.14/) we see the installation screen of the CMS.

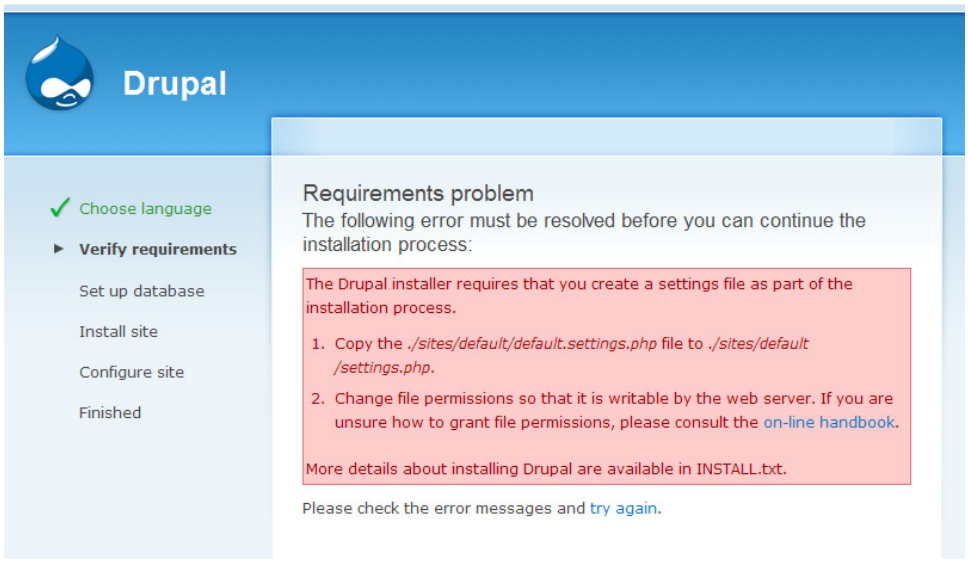


Figure 3.3.1.1: Drupal installation error.

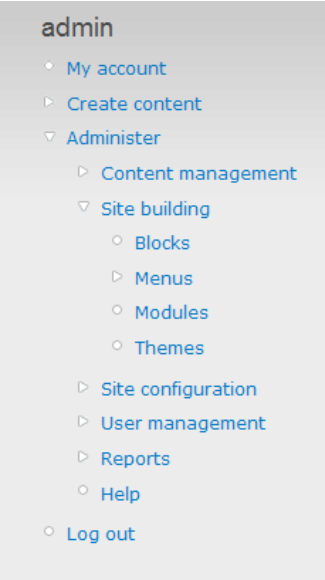
During the installation we get a requirements problem which is an error. What we need to do to fix this is fix problem number one of the screen and copy a file to another folder. Once this is done the problem is fixed and we can continue with the installation process. When using WAMP we get another error at the end of the installation which says that the connection with the mail server has failed. The reason for this is that the settings for the default PHP installation of WAMP are different than the ones required by Drupal. At this moment we are done and can navigate to the already working website. On this page we get some advice on how to change the website to suit our needs.

3.3.2 Usability

When creating a page within the Drupal admin there are not that many things to do. You can choose to add two types of content, namely a Page or a Story. A Page is designed for pages which will not change a lot. For example an "About Us" page or a disclaimer page.

A Story is designed for pages which may change a lot or which users of the website can respond to. This can be a news item or a normal blog. By default a story will be displayed on the home page of the website.

Creating content within the backend is very straightforward and doesn't require much knowledge since the CMS explains itself very easily. One of the downsides is that it doesn't use a WYSIWYG editor. So if you want content with for example a **bold** word in it, you still have to write code for this (bold) which isn't very user friendly in my opinion.



The menu of the backend of the CMS is very easy to understand. You can see a few buttons which contain there sub menus. Changing the users account is one of them. Another one is creating content which exists of stories or pages. The third is the administration section which is the most important part of the backend. Here we can change and delete content, change the websites layout and add or delete modules and themes.

Also set the websites configuration and edit the users which are available within the frontend of the website. Also see any errors or warnings with the website and available updates for the website.

There is also a getting started page on their website which displays more information if needed. This page can be found here:

http://drupal.org/getting-started

Figure 3.3.2.1: Drupal Admin menu.

As we can see in the menu above the CMS works with four types, we have a theme which can include some modules. A module can be a forum or a photo gallery. If we want to display something on the website we have to place this inside a block. So the entire website is build out of blocks and each block contains its own information. For example a forum or photo gallery or maybe a menu for the website. Inside the theme the layout of the blocks and so the entire website can be changed the way you want it to be.

Further the CMS has enough information when working with it and there is a detailed help function available. Here is also more information available about the “blocks” of the CMS.

3.3.3 Search Engine Optimization

Working with SEO urls with Drupal is disabled by default. To enable this we have to go to the configuration part of the website and there the sub section of Clean Urls.

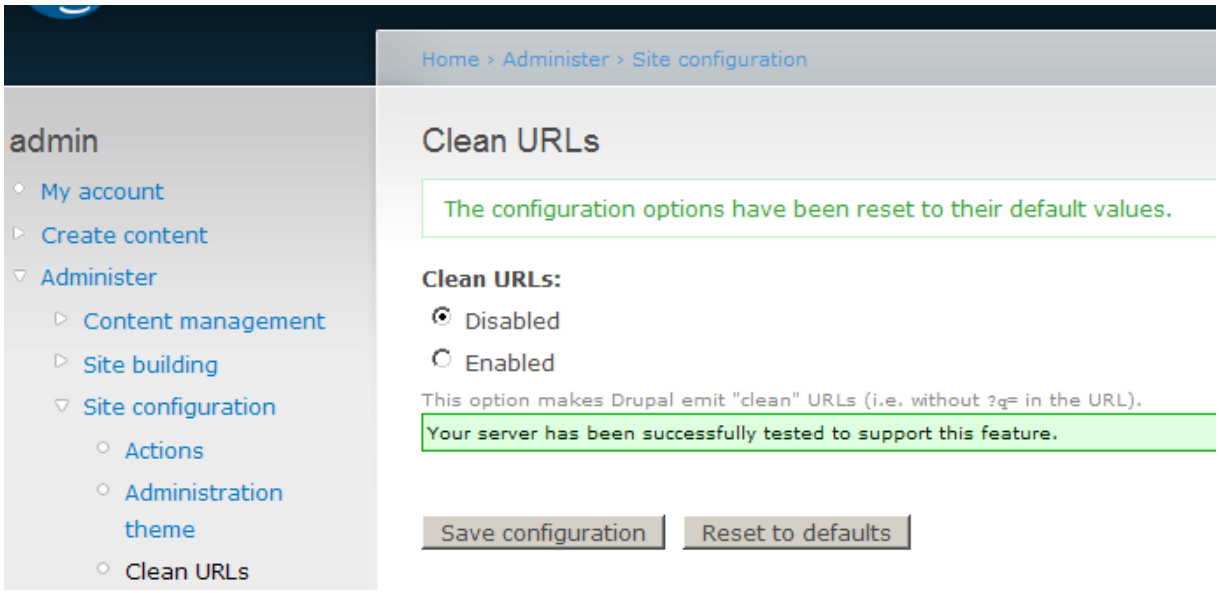


Figure 3.3.3.1: Drupal Clearn Urls.

Here the CMS will automatically test if the configuration of the web server supports working with clean urls, if not there will be a message which says what to do. Usually this means enabling the mod_rewrite module of the apache server.

Then we can enable this function and all the urls will be rewritten to a url which is user friendly and better for search engines. More information about clean urls is displayed on the website of Drupal (24).

3.3.4 Extendability

The Drupal CMS can be expanded with several modules. There are a lot of modules available for download for free. They can be found on the Drupal website (26). There is also a list of all modules in alphabetical order (27).

If you want to write your own module this is also possible. Drupal has a very detailed API which can be used when creating your own module for your website. There is also a getting started page on their website where you can create your module from scratch. These tutorials are separated according to the different versions of the CMS (28). All the information for doing this can be found here:

http://drupal.org/developing/modules

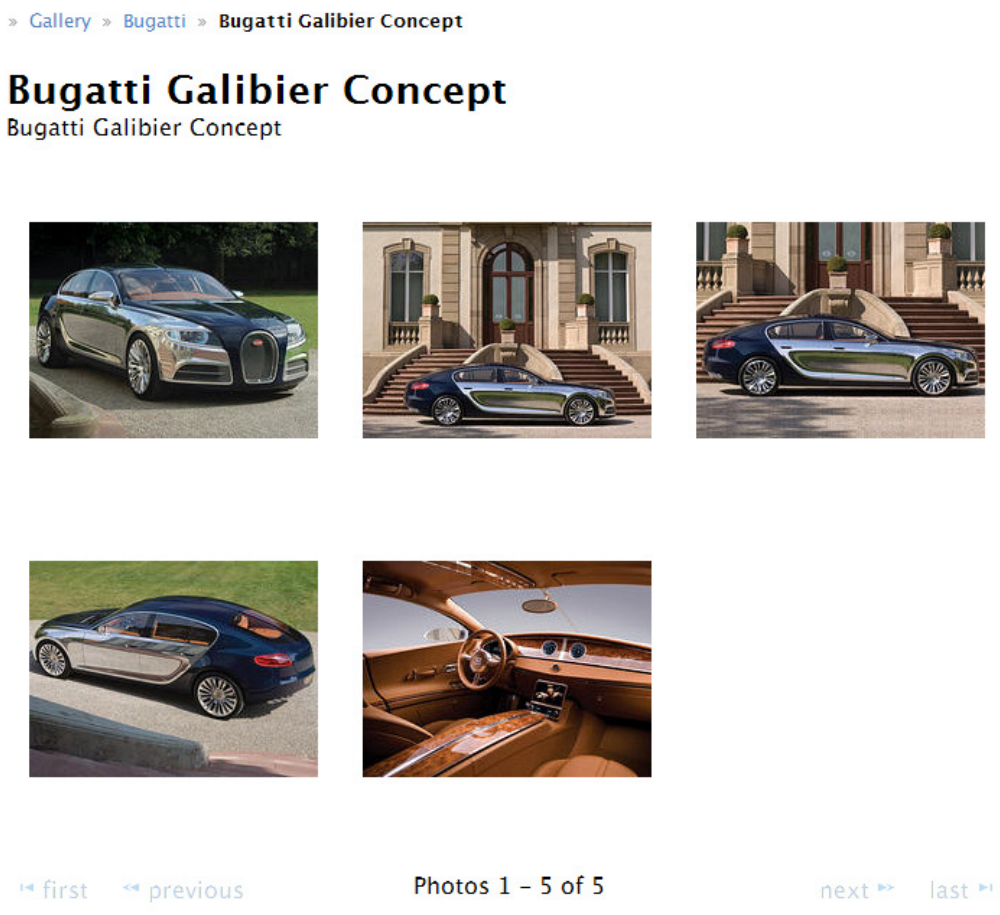


Figure 3.3.4.1: Example module, Photo Gallery, http://www.arabamoto.com/index.php/Bugatti/Bugatti-Galibier-Concept

Above we see an example of a Drupal CMS module namely the Photo Gallery. With this module you can easily create a photo gallery within the Drupal CMS and display your own photos inside it (29).

3.3.5 Templating

With Drupal there are six templates installed by default which can be used. But when you want your website to be different than any other website you want to change your website. On the Drupal website you can download several other templates (30).

If you want to use the default theme but still want to change the layout you can choose to use other colors in the admin panel. There is of course also the possibility to create your own custom style. For this there is also a detailed guide available on the Drupal website (31).

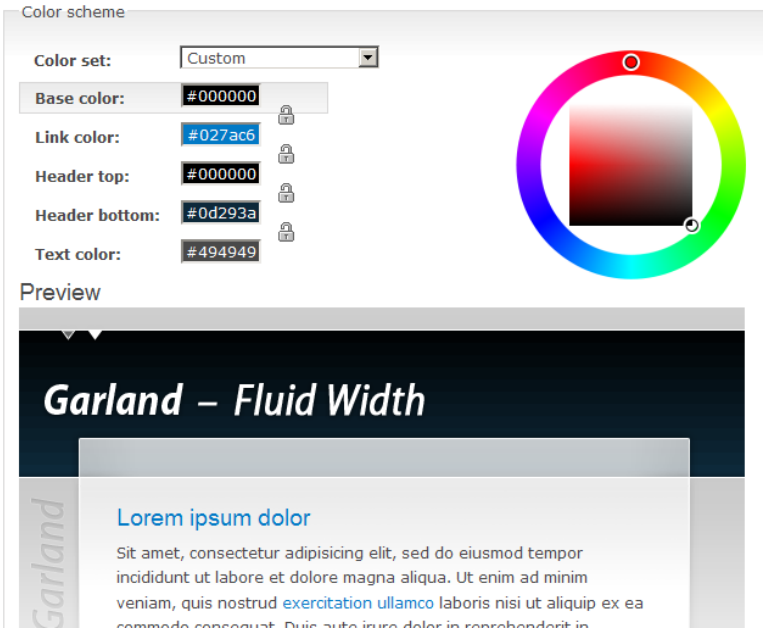


Figure 3.3.5.1: Drupal Theme Color Changer.

3.4 WordPress

What is the history of WordPress? In 2003 a weblog tool was developed which was called b2/cafelog. Later in 2003 the source code of this project was copied to start another project with. The name of this project is called WordPress (32).

WordPress has been rewarded with a *Packt Open Source CMS Award* in 2007. The current version of the CMS is version 2.8.4 and appeared in August 2009 (32).

Figure 3.4.1: WordPress logo, http://upload.wikimedia.org/wikipedia/commons/c/ca/Wordpress-logo.png

3.4.1 Installation

If we want to install WordPress we need to know about the server requirements. After searching a while on their website we can find the requirements and there are not that many compared to the two other CMS systems (33). We do however need to have mod_rewrite enabled which isn't by default within the WAMP web server.

We can download WordPress from their website (34). Here we can choose from a zip or .tar.gz file. Both of these files can be opened and extracted with the most (de-)compression tools. The version I used for this document is the latest version namely version 2.8.4. Working with WordPress also requires a database which is set up. We can do this with the PhpMyAdmin program included inside the WAMP web server.

If this is set up we can navigate to our website which is in our web server. The location for this would probably be like this:

http://localhost/wordpress-2.8.4

Then we will get a message which says there is no configuration file which we need to create. After this we need to set the database configuration and then we can run the installation script. Carefully note the password which is generated during the install and then we are done. When we are logged in we can change all the settings which we have made during the installation.

3.4.2 Usability

Creating a page within the WordPress CMS is very simple. Within the menu we can click on the pages entry, there we can click "Add New". Here we can type the content in the WYSIWYG editor and then click the publish button to finalize the page.

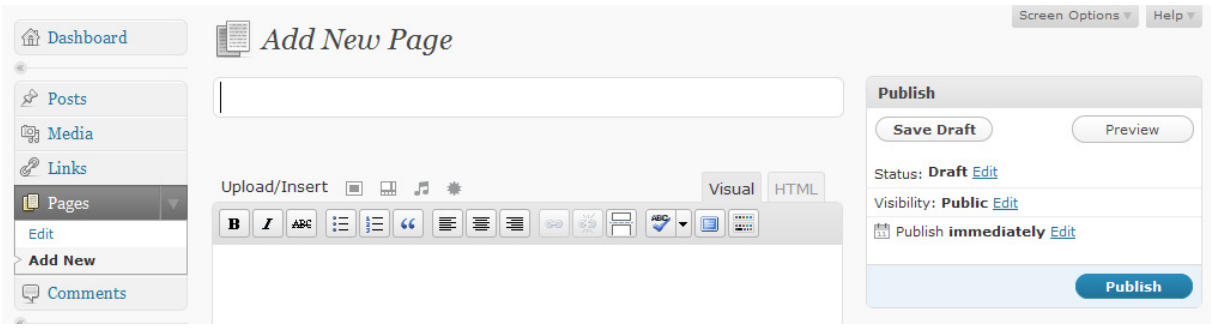


Figure 3.4.2.1: Add new page inside WordPress.

Other things we can do are edit the posts which have been made on the website, manage the links on the website and the media that have been uploaded at the website.

Further does the CMS contain some options for plug-ins and options to change themes. But the CMS is designed for blogging. The things that are included in this CMS work very easily and do not require very advanced knowledge of creating and editing a personal website or CMS's.

3.4.3 Search Engine Optimization

The WordPress CMS requires the mod_rewrite module for apache to be installed. But by default it doesn't use this module at all. And by default user friendly urls as well as search engine friendly urls are disabled. We can enable this in the configuration of the website (35).

If you think this isn't enough to get your website higher in the ranking for search engines there is a complete guide how to optimize your website for search engines. The most of these features are however relevant for most other websites and CMS's. Information for this can be found here:

http://yoast.com/articles/wordpress-seo/

And here:

http://codex.wordpress.org/Search_Engine_Optimization_for_Wordpress

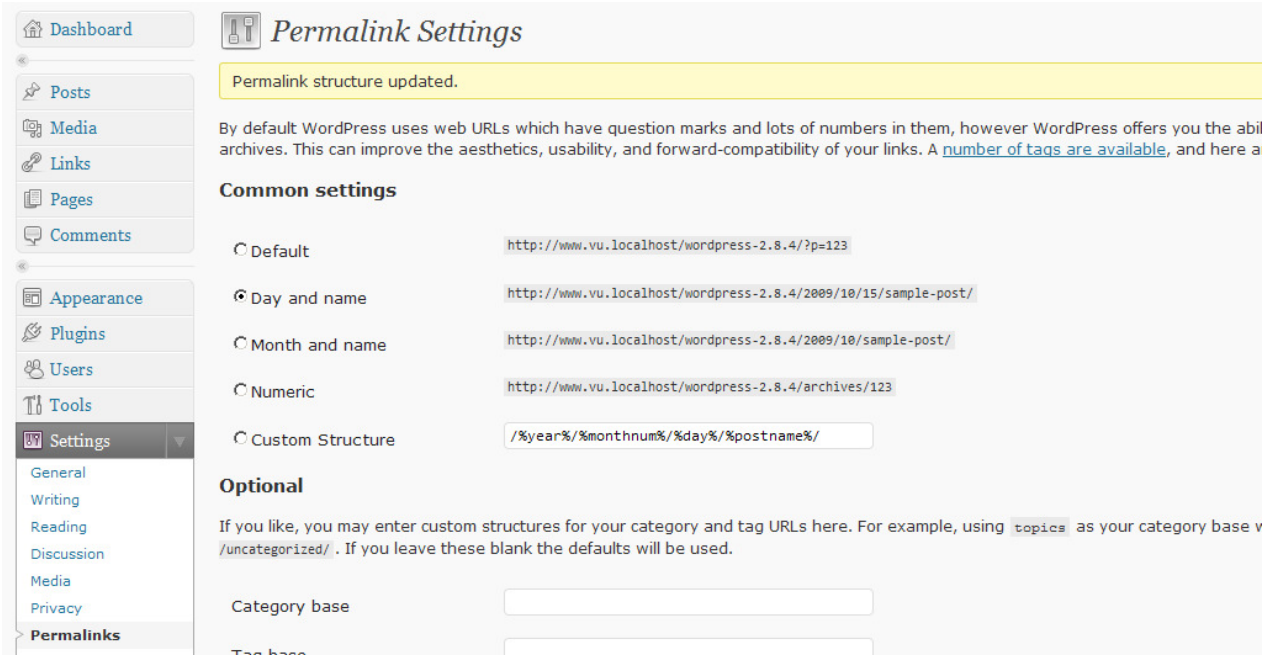
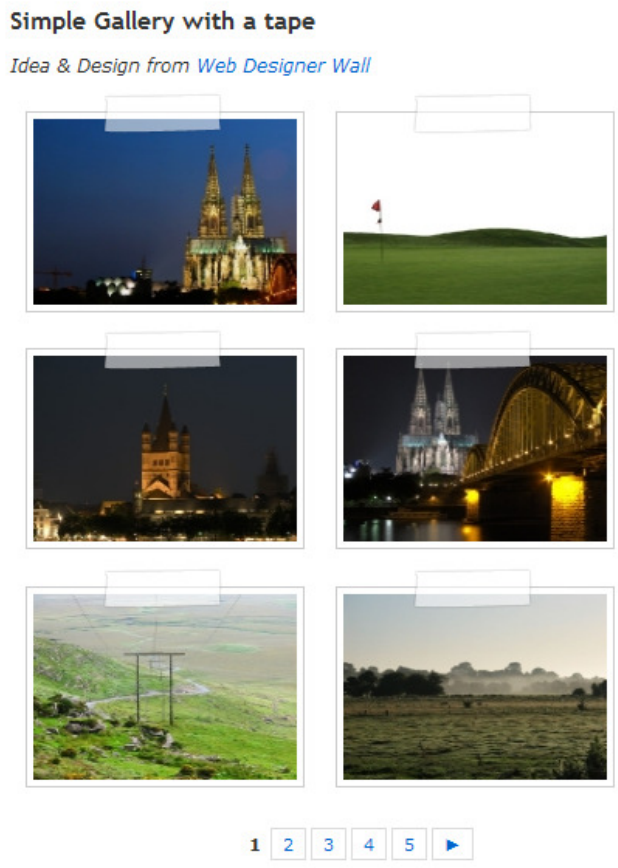


Figure 3.4.3.1: SEO in WordPress.

3.4.4 Extendability

As well as for the other CMS's there are also a lot of plug-ins available for the WordPress CMS. They can be found on the website (36). Some nice features are a photo gallery or a shopping cart.



If you can't find the plug-in which suits your needs you can always program it yourself. This does require knowledge of PHP, CSS and if you want AJAX/JavaScript.

There is a detailed guide with very much information about how the plug-ins should be structured and how they to create a custom plug-in (37). If you want information from the WordPress CMS itself you can use the API to retrieve this information. Information on how to do this can also be found on the website (38).

Figure 3.4.4.1: Plug-in example, Photo Gallery, http://nextgen-gallery.com/templates/example-1/

3.4.5 Templating

Changing the look and feel of a WordPress website can be done in multiple ways. We can add some widgets to change the layout of the website. Also we can change the colors of the headers and change the style sheet of the website to position some elements on a different location. Changing the style sheet does require knowledge of CSS.

Further we can change the theme of the website. By default there are two themes installed. On the website of WordPress we can download many other themes which can be imported into the system (39). If you really want something special you can always write your own theme and import this into the CMS. The WordPress website contains the information on how the themes are structured and how they can be created (40).

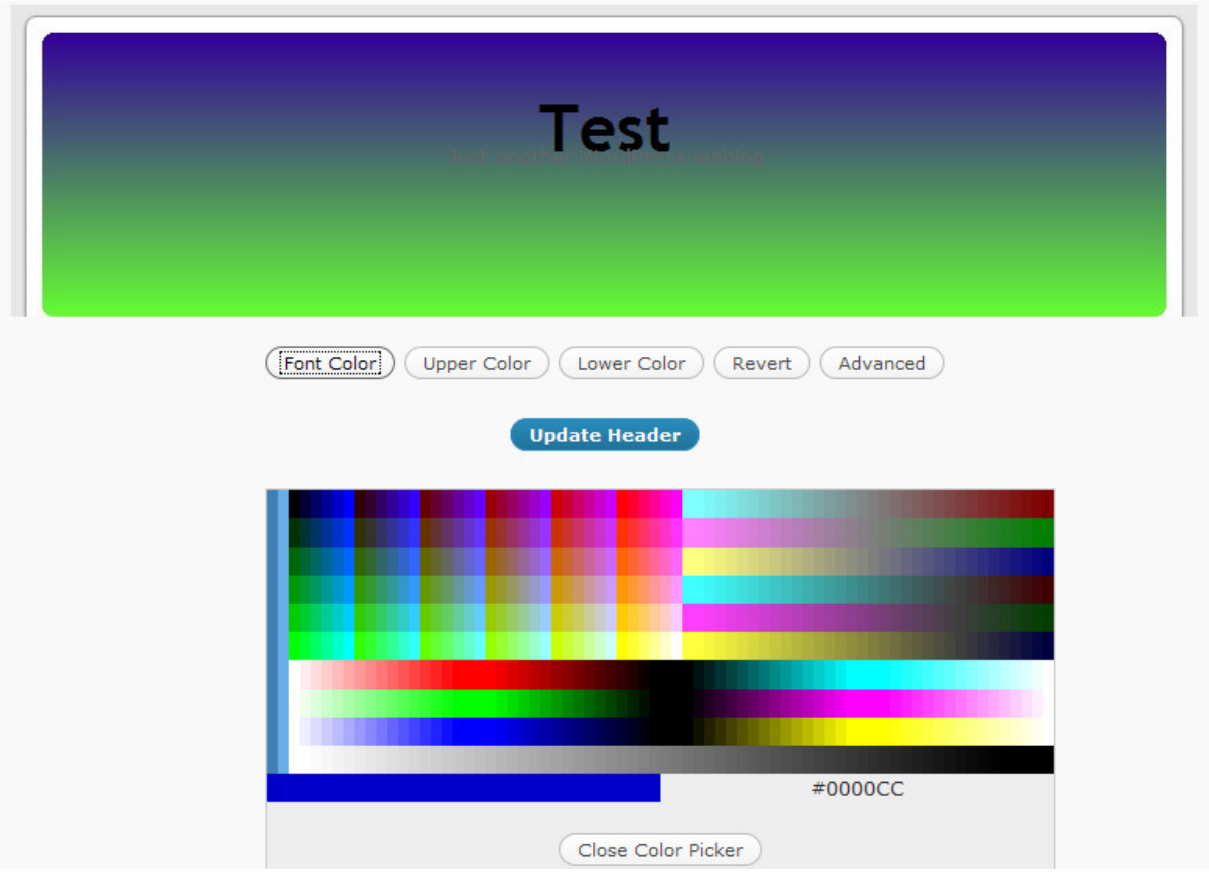


Figure 3.4.5.1: WordPress color picker, changing header.

3.5 Conclusion

After reviewing these three CMS's I will give an overview of there up and downsides. After this is will produce an overview which contains this information. First of all the installation process of the three CMS's. When having WAMP installed the installation process of all three CMS's is relatively the same. We need to set up a database and configure this in the installation process. Further we need to fill in some information about the website.

The usability of the CMS's however is a little different. For Joomla! you really need to read the getting started document before you have a good idea of how the system works and sticks together. Drupal works a lot easier because the structure is better and there is a lot of information available when adding content to the website. WordPress works the easiest if you have no knowledge and haven't read anything of the documentation of the CMS.

Optimizing the CMS's contents for search engines is also relatively equal. We need to enable the mod_rewrite for better indexing with search engines. Further there are a lot of other things we can do as using headings for titles. But all these things can be done with all three systems.

If you want modules for your system to expand it then Joomla! is the best solution. Joomla! has a very large community which shares a large collection of modules. These modules can be written by anyone and therefore might contain security problems. For Drupal and WordPress there are also a lot of modules and extensions available but not as much as for Joomla!. If you want to write your own modules than Drupal is your best bet. Drupal is a CMS with a very detailed API and the source code is very structured and clean.

Creating custom templates is not that easy with any of these systems. If you have no knowledge of doing this, the best solution is to find a custom theme and use this. Otherwise Drupal has a nice color picker which enables you to change the look and feel of the website by simply changing some colors. In my opinion this is something that can be changed in the future by using for example AJAX or Adobe Flex.

The documentation is always another important part when wanting to use a CMS. Since the community of users of Joomla! is very large the most information can be found about this CMS. All three CMS's do have enough information on their website to use the website besides the community. But in my opinion you really have to try if you want to find the information you need on the Drupal website.

3.5.1 Points table

	Joomla!	Drupal	WordPress
Installation	5/5	4/5	5/5
Usability	4/5	4/5	5/5
Search Engine Optimization	5/5	5/5	5/5
Extendibility	5/5	4/5	3/5
Templating	5/5	3/5	4/5
Overall	5/5	4/5	4/5

4 Custom CMS

Comparing and reviewing three well known CMS's was the starting point for developing a custom CMS with the name *JackalCMS*. This first part of this chapter will describe the requirements of the custom CMS and the second part will describe the features of the CMS. These requirements and features are related to the comparison of Joomla!, Drupal and WordPress.

4.1 Requirements Custom CMS

4.1.1 Installation

When distributing a CMS on a large scale, the installation process is important. Since installing the CMS is the first step in using the system this step should give enough confidence to the users to actually use the system.

The custom CMS will not be distributed on a large scale and therefore the installation process is not very important. When decided to use the custom CMS for multiple websites, this should be a matter of duplicating the folder and database and the system will be up and running.

4.1.2 Usability

Usability is very important for a CMS. To improve the usability of a system I have used features which are familiar to an average computer user. The graphical user interface will have the look and feel of the default windows vista control panel.

Further the CMS will be implemented with the use of effects. This gives the application the time to load information from the database without giving the user the feeling he or she has to wait. These details inside the application will improve the usability and therefore the entire CMS.

Another requirement which is connected to the chapter usability is the editor which will be used in order to manage the content of the system. There are multiple solutions which might be feasible, but for this system a *What You See Is What You Get* editor is chosen. The reason for this is that the user doesn't need to have specific knowledge in order to create the content.

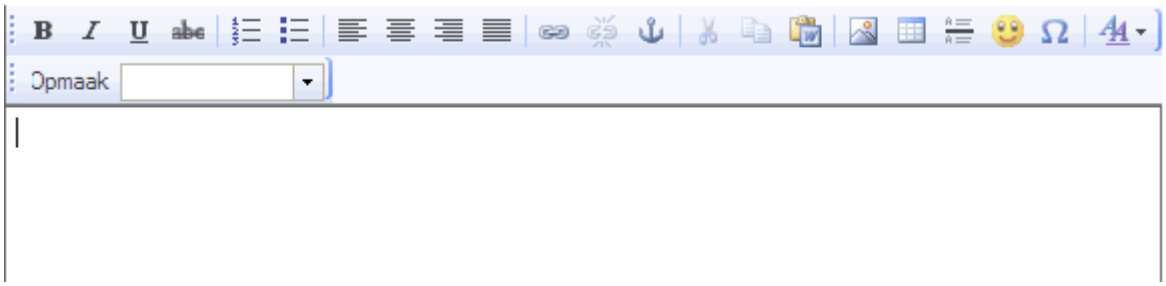


Figure 4.1.2.1: FCKEditor (WYSIWYG editor).

4.1.3 Search Engine Optimization

The reviewed CMS's all have features in order to improve the SEO of the website. One of these features which all three CMS's share is the URL rewriting option. Since this is one of the most important option in order to optimize a website for search engines this is also a requirement for the custom CMS.

The custom CMS will also have the ability to set custom keywords for each page and finally we can create a short description for each page. This information will be displayed in the publishing website which will be index by the search engines which will result in a higher position with search engines.

4.1.4 Extendability

At this moment there are no requirements for extending the CMS. This because the CMS will be created and maintained by myself. The CMS however will be extended in the near future with possibilities to add for example a guestbook and a photo gallery.

The possibility to let developers create custom components for this CMS would require to give a detailed overview of the database scheme which allows the developer to load the data which he requires for his component. Also it would require that the source code of the CMS will be published in order to let the developer include the component since the components need to be compiled into the actual system.

4.1.5 Templating

Changing the layout of the website will in contradiction with the reviewed CMS's not be done via the CMS. The original definition of a CMS says that we are managing the content of a website. This officially does not contain the layout of the website which is publishing the content.

When using Zend and Flex to create a CMS, the entire website is divided into two sections. The first section is the Flex application which is for managing the content of the website and the second section is for displaying the content. Since the second part is created with Zend only, which means PHP in combination with HTML and CSS, we can alter the template of the website by changing the CSS of the website. This means that user doesn't require advances knowledge of the complex code which is behind the website, but that de user only needs basic CSS knowledge in order to change the look and feel of the website.

4.2 Features Custom CMS

4.2.1 General

Features which will be used throughout the entire websites are the use of effects. Also managing and displaying the information in a form which is already familiar to the users is one of the most important parts.

4.2.2 Managing the content

The main feature of a CMS is managing the content of the website. Within this part of the CMS the users can add, edit and delete pages from their website. The structure of the pages will work similar as the windows structure when working with Windows Explorer. This would automatically give users a familiar feeling and therefore makes the system easier to work with.

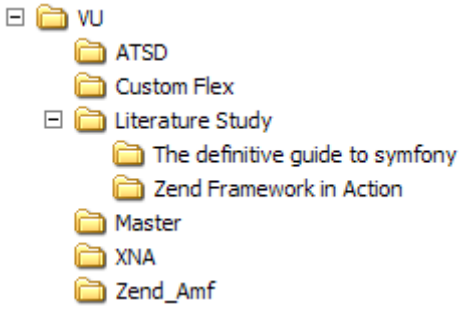


Figure 4.2.2.1 (Left): Windows Explorer tree structure.
Figure 4.2.2.2 (Right): Adobe Flex tree structure.

Within this folder structure deleting or adding a file will be a matter of selecting the right button. Once a file is created the content and specifications can be altered.

4.2.3 Managing the modules

The modules section of the CMS provides the possibility for the user to view the possibilities of the module and at the same time the price for using the module. If the user confirms he/she wants to use this module in his website, this module will be available within the section where the user can alter his pages.

Inside the pages section the user can now include the modules he has purchased. For example a project page where a user can insert multiple images for each page. Each image in his case has a feature that when clicked, the image will be shown in full-size modus.

Other modules which can be included inside this project are a guestbook, photo gallery, forum etc.

4.2.4 Statistics

In the present much attention is paid to see what might be improved in a website. Therefore this will also be included inside the CMS. There are multiple options to receive information about the activities of your website. The first solution is to read the log files of your server and display in a logical way. Doing this is however a very extensive job and requires very much knowledge of the structure of log files. Another downside of doing this is that depending on which server system is used the log files are different. Microsoft's Internet Information Services log files are for instance completely different that the log files which are created by an Apache server. For this reason this will not be included.

The second solution is to include already developed systems into the CMS. This is for this CMS not an option either. The reason not to choose this solution is that they cannot be adjusted to the layout and whishes of the developer/user. Besides this these systems are known to have their flaws which might decrease the user friendliness of the system or worse display incorrect information.

The third solution is using the programming interface which Google offers. Doing so enables users to view the information which Google collects. This information can be used by developers and displayed in the CMS in the desired layout. Since implementing this requires a large amount of time this will implemented in the future and will not be enabled in the current system.

4.2.5 Managing user accounts

The final section of the CMS is the section which if for the management of the users in the system. In order to achieve the best experience for the user this section will use complex effects. For example when searching for a specific user, users which do not meet the requirements will be faded away or faded in if the requirements have changed.

Within this section users can also alter or add users and grant them access to the CMS. Using multiple users for the CMS provides the functionality for logging user actions. For example to see which users altered which page and when the action has been done.

5 Implementation Details

The implementation of the Custom CMS resulted in a list of issues which needed to be solved. This chapter describes these issues, their solutions and their pros and cons.

5.1 Connecting Flex to Zend

The CMS which is build in the environment of Adobe Flex was connected to a backend which is build using the Zend Framework. These environments are completely different and therefore cannot be connected to each other directly. This means there is need for a layer between these two environments. This could be done using multiple solutions. For example using AMF or using SOAP/WSDL requests which are the two most interesting solutions. Let's review the options with their pros and cons starting with WSDL/SOAP.

WSDL/SOAP would be a very suitable solution viewing the current where the usage of WSDL/SOAP has become very popular. WSDL/SOAP uses XML markup to transfer the data in a structured format which can be read by both the Zend Framework (PHP) and Adobe Flex (Action Script). The advantage of this is that the data which is transferred can also be read and interpreted with the human eye. The downside however is that this works with XML and that more information is send across the network than the information you actually want to send. To refine this view the code below.

```
<message name="SayHelloRequest">
   <part name="firstName" type="xsd:string"/>
</message>
<message name="SayHelloResponse">
   <part name="greeting" type="xsd:string"/>
</message>
```

This code shows the information send to the server where we actually only want to do the request below.

```
$response = Server::SayHelloRequest($firstName);
```

As we can notice this causes overhead which will slow down the application with a large factor when the application is expanded, especially when there are a lot of data requests between the Flex application and the Zend backend classes.

The second solution, choosing AMF to transfer the data between Flex and Zend, is very different from WSDL/SOAP. Where SOAP uses XML to transfer the data, AMF uses a binary format. This uses a lot less data transfer on the network and therefore speeds up the process of the requests. There is however one downside, you cannot easily view the data transferred which might result in difficulties when debugging requests from and to the server. Another advantage however is that the entire objects from the Zend backend can be transferred and converted into ActionScript objects which would make it very easy to work with the data. This means reusing the object within Action Script as like it is a normal PHP object.

Delays in an application might result in irritations for the users working with it and therefore I chose for working with AMF to transfer the information between Flex and Zend. Unfortunately during the implementation of this solution one issue occurred. The mapping of PHP objects to ActionScript objects only converts the public variables of the object. During this mapping all functions and private variables are lost which is not the intention. Since one of the most important parts of Object Oriented programming is using private variables and functions to get the information this is essential.

When searching for a workaround I came up with the solution of using wrappers in combination with serializing the object. The serialization saves the state within Flex. This makes it possible to work with the object when returning it back to the Zend Framework. I will not go into the details of this since this is technical information.

5.2 Flex WYSIWYG editor

Adobe Flex offers a default editor which can create HTML code. This component would be ideal for the purpose of a CMS. This component called "RichTextEditor" within the Adobe Flex SDK unfortunately has limitations.

Figure 4.2.1: Flex RichTextEditor, http://livedocs.adobe.com/flex/gumbo/html/images/tc_AboutRichText001.png

When we look at figure 4.2.1 we see that the options which are available are the most general options which are needed when editing rich content. However this editor lacks functionalities to for example insert an image.

During the implementation of the Custom CMS I tried to extend the default component in order to add the functionality for inserting images, but due to some limitations to the HTML engine and the Flex SDK there still are problems which cannot be worked around. Therefore this component is not used for the implementation of the Custom CMS.

To manage the content I worked around this problem I used a JavaScript WYSIWYG editor inside the Flex application. Doing this enables working with images inside the HTML and outputs valid HTML which isn't the case with the Flex RichTextEditor. Below we find a piece of code produced by the Flex RichTextEditor.

```
This is <b> Bold </b> text.

<TEXTFORMAT LEADING="2" >
        <P ALIGN="LEFT">
                <FONT FACE="Verdana" SIZE="10" COLOR="#0B333C" LETTERSPACING="0" KERNING="0">
                        This is <b> Bold </b >text.
                </FONT>
        </P>
</TEXTFORMAT>
```

Reviewing this piece of code we notice that this is not valid XHTML and that well known browsers will not interpret it as mentioned. This where a JavaScript WYSIWYG editor would produce the following HTML code which is XHTML valid.

```
<p align="LEFT">
        This is <b> Bold </b> text.
</p>
```

By default browsers render Flash content in front of normal HTML. Since the JavaScript WYSIWYG editor is displayed within HTML this in its place needed to be placed in front of the Flex Application in order for the editor to work. This required that the configuration of the application needed to be altered to support the use of HTML in front of the Flex application.. Secondly the editor makes use of an iframe which is loaded in the background of the Flex Application. When a page is selected for editing, the iframe is made visible and shown in front of the application. By saving the page the iframe will submit the data to the backend which in his turn saves the information in the database. If this process executed successfully the iframe will be hidden and a message will be displayed that the information is successfully saved.

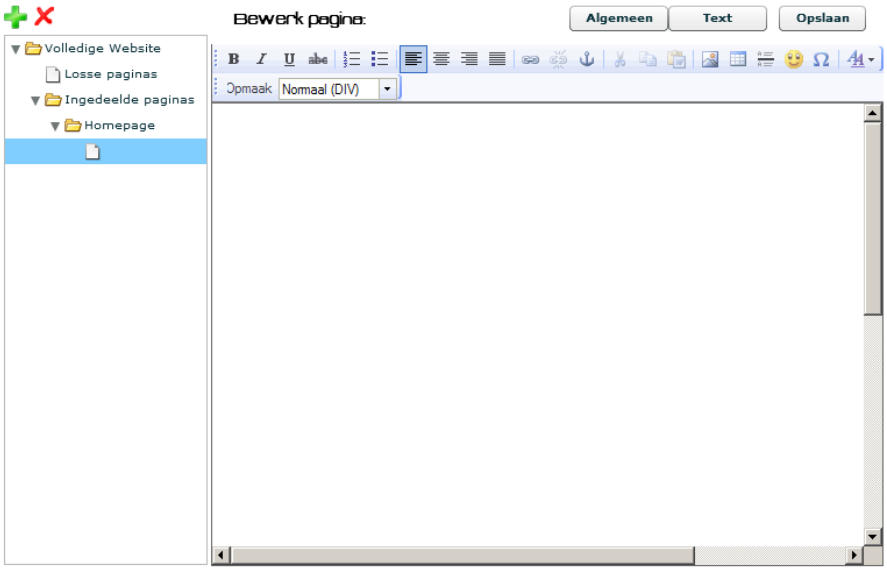


Figure 4.2.2: Jackal CMS editor with FCKEditor.

5.3 Why FCKEditor

There are several different "what you see is what you get" editors with each having their specific features. Since the Flex editor doesn't suit the needs we need to select another editor. There are two larger editors which are TinyMCE and FCKEditor. FCKEditor already has a newer version called CKEditor. When working on my custom CMS CKEditor was in development phase and therefore CKEditor was not an option. Thus my choice was between TinyMCE and FCKEditor. For a decent choice here is a comparison between these two editors.

	FCKEditor	TinyMCE
Output Code:	~~Very Ugly and invalid Markup~~ With the 2.6 RC update the code is valid.	Uses valid XHTML and CSS code
Installation Connectors:	FCKEditor is able to install with ASP.Net, ASP, ColdFusion, PHP, Java, Active-FoxPro, Lasso, Perl, Python	TinyMCE is able to install with PHP, .NET, JSP, Coldfusion
Upload:	FCKEditor allows you to upload without any plugins.	TinyMCE doesn't allow upload, but you can with a plugin that you have to buy.
Image Management:	FCKEditor allows you to insert images from the server that you uploaded but without CKFinder you can't delete images.	TinyMCE doesn't allow image management but there is a plugin that you pay for.
Load Time:	Slower than TinyMCE	Faster than FCKEditor
Browser Support:	Internet Explorer 5.5+, FireFox 1.5+, Safari 30+, Opera 9.0+, Netscape 7.1+, Camino 1.0+	Internet Explorer 5.5+, FireFox 1.5+, Safari 3, Opera 9.0+, Camino 1.0, SeaMonkey 1.0.5
Installation:	The Installation for FCKEditor is fairly easy. The only problems I have had was trying to get the right directory.	The installation for TinyMCE was very easy. I had no problems.
Customization:	FCKEditor is much harder to add customizable plugins, but you are able to customize the theme and the toolbar.	TinyMCE is fully customizable through themes and plugins.

Table 5.3.1 Origin: http://webtecker.com/2008/04/02/fckeditor-vs-tinymce/

As mentioned the table above originates from the webtecker website. My conclusion is based on this information and my experience with TinyMCE and FCKEditor. Since one very important part of the editor is interacting with images FCKEditor is already in the advantage. The second reason for choosing FCKEditor is that it also produces valid XHTML and works with PHP.

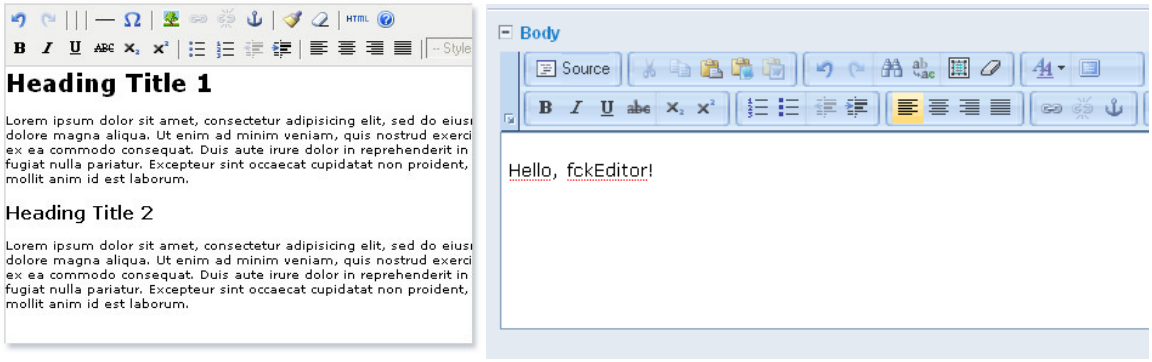


Figure 5.3.1 (left): TinyMCE editor (http://www.thisisblend.com/assets/images/tinymce.jpg)
Figure 5.3.2(right): FCKEditor (http://mwoa.ru/images/uploads/fckeditor.gif)

5.3.1 Configuring FCKEditor

The implementation of FCKEditor is very straightforward. There are a few important issues which should be configured correctly in order to let FCKEditor work with the custom CMS. The first thing to do is copy the source of the FCKEditor downloaded from their website into a folder within the web-server. Once this is done we need to set up the connection between FCKEditor and Zend. Since FCKEditor is build with JavaScript we need to set a connector file which is included inside the editors' source. The file is located at the following location:

```
fckeditor\editor\filemanager\connectors\php\config.php
```

The following files need to be set with the correct values.

```
$Config['Enabled'] = true ;
$Config['UserFilesPath'] = 'http://www.urlofwebsite.com/editor-data/';
$Config['UserFilesAbsolutePath'] = 'c:\dir-to-www\editor-data\';
```

At this moment including the editor inside the code is possible. The 'UserFilesAbsolutePath' above is the folder where the images are stored when being uploaded. The 'UserFilesPath' is the location where the images are being loaded from.

The second step in configuring the editor is minimizing the buttons which are displayed in the toolbar of the editor. Since the custom CMS is created to provide a simple and user-friendly interface the default toolbar configuration of FCKEditor doesn't suit the needs. This can be altered by using a custom configuration file which includes a toolbar set for the editor. Below we find the custom configuration which I included in the CMS.

```
FCKConfig.ToolbarSets["JackalToolbar"] = [
        ['Bold','Italic','Underline','StrikeThrough',
        '-','OrderedList','UnorderedList',
        '-','JustifyLeft','JustifyCenter','JustifyRight','JustifyFull',
        '-','Link','Unlink','Anchor',
        '-','Cut','Copy','PasteWord',
        '-','Image','Table','Rule','Smiley','SpecialChar',
```

```
        '-', 'TextColor'],
        ['FontFormat']
    ] ;

    FCKConfig.ToolbarCanCollapse = false;
```

Using this configuration file will minimize the number of buttons and will feel pleasant for the users. The last line in the configuration file disables the possibility to collapse the toolbar. The reason to set this is to prevent misunderstandings for the user and a better feeling. This also gives the idea of working with Microsoft Office to the user of the editor.

6 Evaluation

The evaluation section contains three sections, the evaluation of the tutorial, the evaluation of Adobe Flex for CMS's and the evaluation of the Zend Framework for CMS's.

6.1 Tutorial

The tutorial shows that Adobe Flex in combination with the Zend Framework is a nice solution for creating a CMS. With Flex you can create applications which work very user friendly. Another nice feature of Flex is that with the user of the default components you can create applications which have a very consistent layout.

For the developer it is relatively easily to implement a CMS with Flex and Zend. There are however some downsides. For example the RichTextEditor from Flex which by default doesn't support possibilities to insert images into the editor.

Creating the publishing website with Flex is in my opinion not a good solution. The reason I have this opinion is that it is very difficult to have your application adapted to multiple resolutions. I also think that sending many requests to the server would be needed for complex websites. This would result in a heavy server load and might also make the website a lot slower as when the website would be build with for example just the Zend Framework.

Another reason why it is better to not use a full Flex website is that search engines like Google are not good in indexing websites these websites. This because a Flex project is compiled into one single file which doesn't contain information like keywords for each page or have links to other pages which search engines can follow.

6.2 Adobe Flex

Before starting with my final project I had some experience with Adobe Flex. Adobe Flex is very suitable for building a CMS. It has the possibilities to build a folder structure which is familiar for the most computer users. This can be used for displaying the pages of a website.

Also the effects can create a very good look and feel for the end-user. For example displaying an effect when information needs to be loaded from the database can hide the delay and therefore makes it more pleasant for the user. Also large changes in the application can be covered up by using effects. For example the large change when changing from a login screen to the actual system can be made a lot smoother by using fading effects.

The final nice feature of Flex is that when loading the application all the information can be loaded from the database. This way there won't be any other delays in the application later on. Doing this however might resolve into problems if the data loaded will change a lot or can be changed by multiple users.

6.3 Zend Framework

Working with the Zend Framework requires knowledge from PHP. Before this final project I didn't have much experience with the Zend Framework. It is however not to difficult to work with the Zend Framework once you know how to build a website with Zend. The framework has some very nice features which makes working with the framework a lot faster than when you should build all the

components yourself. To give one example which is used inside the tutorial, we used the Amf module of the framework.

Another big advantage is the fact that a website with Zend is very structured by default. This makes it a lot easier to reuse your own code. Also debugging your website is a lot easier because you know exactly where to search for your bugs and errors.

For the tutorial the only module of the Zend Framework used was the Amf module, however when extending the website and implementing more complex functionality Zend provides enough features for the programmer to make it as easy as possible.

7 Conclusion

At the moment of writing this document Adobe Flex has been used as programming language for many web applications. Currently Adobe has released the fourth version and the community behind Flex is still growing. For the Zend Framework this is similar. While Content Managing has been used for a long time there aren't any CMS's which are developed with Flex. This thesis discusses the reasons to develop a CMS for yourself, the important sides and compares already existing CMS's. Further it includes a tutorial which allows the reader to implement his own Flex CMS. Next to this it discusses the CMS which I have developed with their features, issues and problems which I ran into.

Comparing the currently existing CMS's showed that you should state your requirements before developing your own CMS. This means you should know what you want to do and what will it be used for. Reviewing and comparing existing CMS's showed me that a CMS should be very user friendly and should work without using any manual or help functionality. If you should use this than working of the system is too complex. Important features of a CMS are Search Engine Optimization (SEO) and speed. SEO for higher results of the website in search engines. The speed for user friendliness.

Building a CMS with Adobe Flex seems a very good option. Mainly because Flex can interact with the user as a real desktop application which actually runs in a browser. This gives the user of the system the advantage of better speed (loading the data when the time is at best) and using options which are known from for example the windows environment. This includes drag and drop functionality and if necessary there are options to implement a trash can.

Using Flex for the publishing website is not a very good option because Flex is not very well indexed by search engines. It is however possible to insert small widgets which are developed with Flex. One good example for this would be a small chat. Using Flex with Zend has the advantage that Zend includes a large number of modules which can then also be used inside the publishing website. Examples of this are using the Google data API or for example displaying twitter messages on the website.

The tutorial shows that your do need to have knowledge of Flex and PHP. Depending on how complex the developer wants to build his system the level of knowledge also varies.

8 Epilogue

Personally I think that creating a CMS with Flex and Zend is a very good starting point. However I also think that AJAX might also be a good option for building CMS's. I intend to explore the options for creating a CMS with AJAX in the near future.

I would like to thank prof. dr. Anton Eliëns for putting me back on the right track after the first version. Also for giving me good information about how to do this project and encouraging me to write a tutorial which should be readable and understandable by anyone.

Further I would like to thank Rahul Premraj for taking over the position of second reader after Ronald Siebes mentioned he didn't have the time to be the second reader for the second version of this document.

Creating a CMS on my own was a very interesting experience which also learned me to connect different programming languages together and figure out what problems can arise and to be creative and look for solutions and workarounds.

References

1. Rich Internet Application. *Wikipedia.org.* [Online] [Citaat van: 12 Januari 2010.] http://nl.wikipedia.org/wiki/Rich_Internet_Application.

2. Content Management System. *Wikipedia.org.* [Online] [Citaat van: 12 Januari 2010.] http://en.wikipedia.org/wiki/Content_management_system.

3. Framework. *Wikipedia.org.* [Online] [Citaat van: 2009 November 3.] http://en.wikipedia.org/wiki/Framework.

4. Adobe Flash Player PC Penetration. *Adobe.com.* [Online] [Citaat van: 11 Februari 2010.] http://www.adobe.com/products/player_census/flashplayer/PC.html.

5. Adobe to acquire Macromedia. *Adobe.com.* [Online] [Citaat van: 2009 November 3.] http://www.adobe.com/aboutadobe/invrelations/adobeandmacromedia.html.

6. Adobe Flex. *Wikipedia.org.* [Online] [Citaat van: 2009 November 3.] http://en.wikipedia.org/wiki/Adobe_Flex.

7. **Zafra, Arnold.** Top 40 best free CMS. *blorge.com.* [Online] September 1, 2008. [Cited: September 22, 2009.] http://tech.blorge.com/Structure:%20/2008/09/01/top-40-open-source-content-management-systems-cms/.

8. **Scott, Jeffrey.** The Top 10 Open Source Content Management Systems. *WebDevNews.com.* [Online] 1 September 2008. [Citaat van: 21 September 2009.] http://webdevnews.net/2008/09/the-top-10-open-source-content-management-systems/.

9. Mambo (software). *Wikipedia.org.* [Online] [Citaat van: 23 September 2009.] http://en.wikipedia.org/wiki/Mambo_(software).

10. Joomla. *Wikipedia.org.* [Online] [Citaat van: 23 Semptember 2009.] http://en.wikipedia.org/wiki/Joomla.

11. Nominate the Best Open Source PHP CMS Winner. *PacktPub.* [Online] [Citaat van: 23 September 2009.] http://www.packtpub.com/nominate-best-open-source-php-cms.

12. Joomla Technical Requirements. *Joomla.org.* [Online] [Citaat van: 23 September 2009.] http://www.joomla.org/technical-requirements.html.

13. Apache, MySQL, PHP pour Windows. *WampServer.* [Online] [Citaat van: 23 September 2009.] http://www.wampserver.com/.

14. Download Joomla! *Joomla.org.* [Online] [Citaat van: 23 September 2009.] http://www.joomla.org/download.html.

15. How to enable SEF in Joomla 1.5. *SiteGround.com.* [Online] [Citaat van: 30 September 2009.] http://kb.siteground.com/article/How_to_enable_SEF_in_Joomla_15.html.

16. Advanced options in Joomla 1.5: SEF URLs. *SiteGround.com.* [Online] [Citaat van: 30 September 2009.] http://www.siteground.com/tutorials/joomla15/joomla_seo.htm.

17. Joomla 1.5 SEO Patch. *extensions.joomla.org.* [Online] 21 December 2007. [Citaat van: 30 September 2009.] http://extensions.joomla.org/extensions/site-management/seo-a-metadata/3778.

18. Browse Extensions by Categories. *extensions.joomla.org.* [Online] [Citaat van: 30 September 2009.] http://extensions.joomla.org/extensions.

19. Joomla: Simple tutorial how to make your own component . *motov.net.* [Online] 18 October 2007. [Citaat van: 30 September 2009.] http://www.motov.net/joomla-simple-tutorial-how-to-make-your-own-component.html.

20. Joomla 1.5 Templates. *SiteGround.com.* [Online] [Citaat van: 30 September 2009.] http://www.siteground.com/joomla-hosting/joomla-templates.htm.

21. How to Create a Basic Template. *HowToJoomla.net.* [Online] 25 September 2006. [Citaat van: 30 September 2009.] http://www.howtojoomla.net/2006092520/how-tos/templates/how-to-create-a-basic-template.

22. Drupal. *nl.wikipedia.org.* [Online] [Citaat van: 30 September 2009.] http://nl.wikipedia.org/wiki/Drupal.

23. System Requirements. *Drupal.org.* [Online] [Citaat van: 9 October 2009.] http://drupal.org/requirements.

24. Drupal. *Drupal.org.* [Online] 28 September 2003. [Citaat van: 9 October 2009.] http://drupal.org/project/drupal.

25. Clean URLs. *Drupal.org.* [Online] 3 September 2009. [Citaat van: 12 October 2009.] http://drupal.org/node/15365.

26. Modules. *Drupal.org.* [Online] [Citaat van: 12 October 2009.] http://drupal.org/project/Modules.

27. Alphabetical list of Modules, by core version. *Drupal.org.* [Online] 19 May 2009. [Citaat van: 12 October 2009.] http://drupal.org/node/206666.

28. Module developer's guide. *Drupal.org.* [Online] 13 July 2009. [Citaat van: 12 October 2009.] http://drupal.org/developing/modules.

29. Gallery. *Drupal.org.* [Online] 28 September 2003. [Citaat van: 12 October 2009.] http://drupal.org/project/gallery.

30. Themes. *Drupal.org.* [Online] [Citaat van: 12 October 2009.] http://drupal.org/project/Themes.

31. Drupal 6 Theme Guide. *Drupal.org.* [Online] 14 Augustus 2009. [Citaat van: 12 October 2009.] http://drupal.org/theme-guide/6.

32. WordPress. *Wikipedia.org.* [Online] [Citaat van: 13 October 2009.] http://en.wikipedia.org/wiki/Wordpress.

33. Requirements. *Wordpress.org.* [Online] [Citaat van: 13 October 2009.] http://wordpress.org/about/requirements/.

34. Download WordPress. *Wordpress.org.* [Online] [Citaat van: 13 October 2009.] http://wordpress.org/download/.

35. Clean URL's with Wordpress. *TeamTutorials.com.* [Online] 28 March 2007. [Citaat van: 15 October 2009.] http://teamtutorials.com/web-development-tutorials/clean-url%E2%80%99s-with-wordpress.

36. Plugin Directory. *WordPress.org.* [Online] [Citaat van: 15 October 2009.] http://wordpress.org/extend/plugins/.

37. Writing a Plugin. [Online] [Citaat van: 15 October 2009.] http://codex.wordpress.org/Writing_a_Plugin.

38. Plugin API. *WordPress.org.* [Online] [Citaat van: 15 October 2009.] http://codex.wordpress.org/Plugin_API.

39. Free Theme Directory. *WordPress.org.* [Online] [Citaat van: 15 October 2009.] http://wordpress.org/extend/themes/.

40. Theme Development. *WordPress.org.* [Online] [Citaat van: 15 October 2009.] http://codex.wordpress.org/Theme_Development.

41. Zend Framework Quick Start. *framework.zend.com.* [Online] [Citaat van: 2009 November 13.] http://framework.zend.com/docs/quickstart/create-your-project.

42. Flex Quick Start: Getting Started. *Adobe.com.* [Online] [Citaat van: 2009 November 17.] http://www.adobe.com/devnet/flex/quickstart/coding_with_mxml_and_actionscript/.

1 Appendix: Tutorial

1.1 Introduction

Within this tutorial we will create a CMS which is build with three different programming languages, namely Action Script, MXML (Adobe Flex) and PHP (the Zend Framework).

Within the requirements section we will see which language will be used for which purpose. Also we will see what will and what won't be implemented.

Knowledge of Object Oriented PHP programming is required for this tutorial. Some basic knowledge of Apache and MySQL would be nice but isn't required. The Flex part of this tutorial is very basic and knowledge about Flex would be nice but is not required.

1.2 Requirements Flex CMS

For the CMS which we will build in the tutorial there are some requirements. Namely the frontend of the CMS will be build with Adobe Flex where the backend of the CMS will be build with the Zend Framework.

For the transfer of the data between Flex and Zend we will use the Amf protocol which is implemented in a module of Zend and a standard feature of the Flash player.

The user should be able to create pages for his website and change the content. For each page he or she is allowed to add one image. This image can be uploaded or the location can be given. Changing the content on the page can be done with a WYSIWYG editor. The pages Can also be deleted if necessary.

Once all the content is created we will also create a website which displays the content. This website will be build using the Zend Framework standalone. This means the publishing website doesn't use Adobe Flex.

As a nice feature I will include SEO url's inside the publishing website. For example the page with the title *"General Information"* will be called with the url: http://localhost/get/page/name/General-Information/. This is better for search engines and also friendly for the human eye.

Further the user can enter extra information for the pages which he has created. For instance the keywords and the description for the page. These things also improve the ranking in search engines.

The system will be build with a WAMP (Windows Apache MySQL PHP) environment. For this tutorial Adobe Flex 3 will be used instead of Flex 4 since version 3 is better known and doesn't require that much time to get familiar to where Flex 4 has a completely different structure for the programmer.

Figure 9.2.1: Wamp installation wizard, http://img.mywindowspc.com/wg/0902/27wamp/wamp10.png

1.3 Installing the used software

The WAMP web server can be downloaded at the following URL:

http://www.wampserver.com/en/download.php

I will leave the installing process to the user since this falls outside the scope of this tutorial. Therefore I will continue with the set up of the Zend Framework. The framework can be downloaded from the following URL:

http://www.zend.com/community/downloads

For this tutorial we will use the full framework release. Select the zip file and download it. Once downloaded open it with an (de)-compression tool like WinZip or 7Zip. The download does require you to register. The version I used for this tutorial is 1.9.5. Inside the zip file we find a folder called *"ZendFramework-1.9.5".* Inside this folder there is a folder called *"library",* this is the folder which we need. Normally WAMP is installed into the folder c:\wamp. The folder which is viewable from the browser is then c:\wamp\www. The website which we will be creating also has a default structure. More information about the structure is also available on the Zend website (38).

We can choose to generate the folder structure of the project automatically or manually. We will choose to generate the project. To do this we need to extract the library folder and the bin folder inside the zip file to the root of the web server (c:\wamp\www).

Then we need to open a command window in windows. To do this: Start>Run and type cmd. Then press enter. The following screen will be opened:

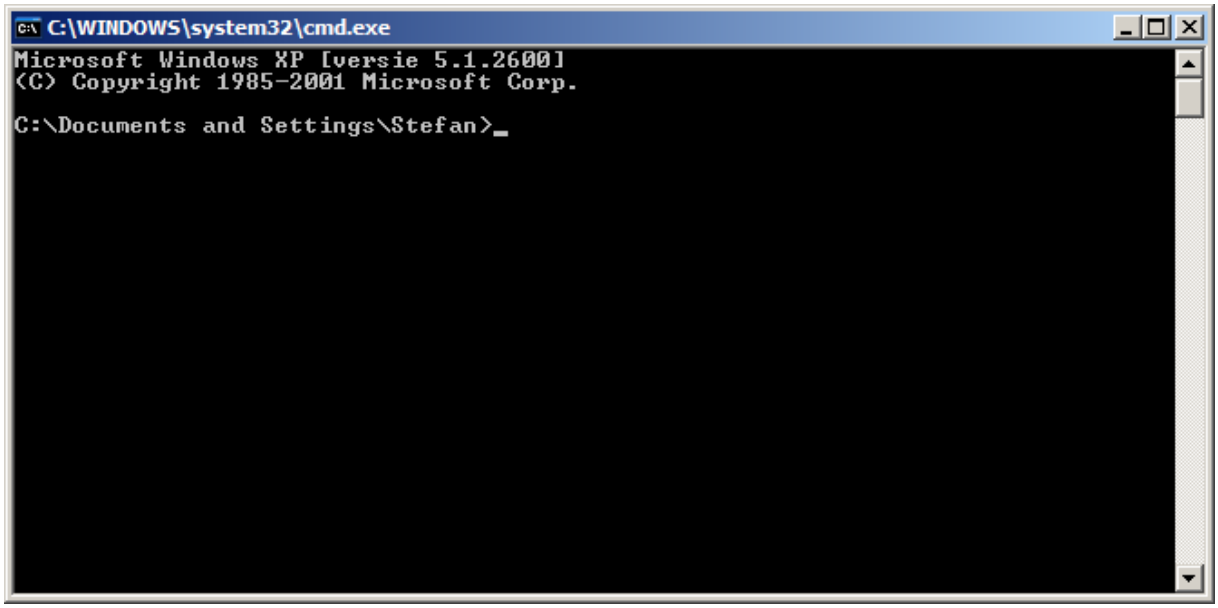


Figure 9.3.1: Command window.

Then we go to the folder of the binary folder of Zend inside the web server by typing the following:

Cd c:\wamp\www\bin\

We can now create a new project with the following command:

Zf create project ../tutorial

Doing this might result in the following error:

"php.exe" wordt niet herkend als een interne of externe opdracht, programma of batchbestand.

This error can be resolved by setting the system path to add the folder where PHP is installed. The command for this is:

path=%PATH%;c:\wamp\bin\php\php5.2.8

If we type the project creation command again everything should work. We can see that there is a new folder created in the root of the web server called tutorial. If this is the case we can delete the bin and library folder in the root of the server and close the command window.

Once this is done the structure of the tutorial folder can be viewed below.

```
Tutorial
|-- application
|   |-- Bootstrap.php
|   |-- configs
|   |   `-- application.ini
|   |-- controllers
|   |   |-- ErrorController.php
|   |   `-- IndexController.php
|   |-- models
|   `-- views
```

```
|       |-- helpers
|       `-- scripts
|           |-- error
|           |   `-- error.phtml
|           `-- index
|               `-- index.phtml
|-- library
|-- public
    `-- index.php
```

Due to the generation of the project in a different location of where the library was originally placed we the library folder might be empty. To resolve this we might need to place the contents of the library folder of the zip file into the library folder inside the tutorial folder.

At this point we are ready to see if the project is created successfully and is working. We can navigate to the URL of the project and see the result.

http://localhost/tutorial/public/

The result of this should be the same as the following image. If this is the case we know for sure that the installation of the Zend Framework was a success.

Figure 9.3.2: Result of new created Zend project.

If the requests results in a internal server error this might mean that the required apache module "mod_rewrite" is disabled. We can enable this by navigating to the following folder:

C:\wamp\bin\apache\Apache2.2.11\conf

Within this folder we find the a file called ” httpd.conf”. Here we need to change the following line:

#LoadModule rewrite_module modules/mod_rewrite.so

Into:

LoadModule rewrite_module modules/mod_rewrite.so

After this we need to restart the apache server. This can be done by clicking on the icon in the right side of the windows menu bar and selecting restart all services. This should resolve the internal server error.

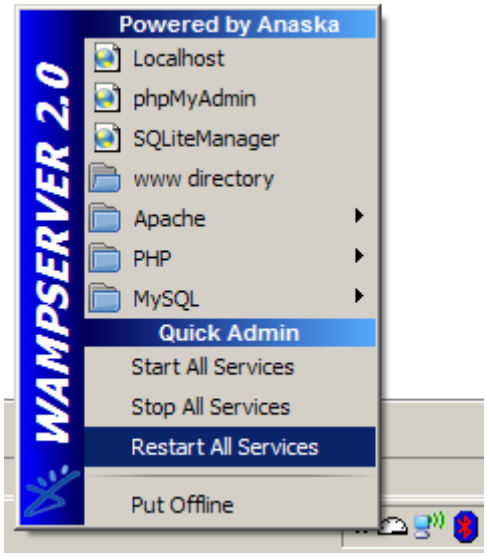


Figure 9.3.3: WAMP menu, restart apache.

Now the set up of the Zend Framework is done, we can continue with the location of the Flex SDK. For this tutorial the idea is to use the SDK without the Flex/Flash Builder. This means that we will use the command line tool from the SDK to compile the source code which we have written. The advantage of this is that we don’t need any licenses.

The latest version of the Flex SDK can be found on the following website.

http://www.adobe.com/cfusion/entitlement/index.cfm?e=flex3sdk

Once the SDK is downloaded we will extract it to the following location.

C:\wamp\www\tutorial\sdk

Lets now create a simple example to see if everything is working correctly. Create a file called tutorial.mxml at the location “c:\wamp\www\tutorial”. Paste the following code inside the file:

```
<?xml version="1.0" encoding="utf-8"?>
    <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" horizontalAlign="center"
                    verticalAlign="center">
    <mx:Script><![CDATA[
            import mx.controls.Alert;
    ]]></mx:Script>
    <mx:Button id="myButton" label="I'm a button!" click="{Alert.show('Hello World!');}" />
</mx:Application>
```

This is a very simple Flex application which has a button with an click handler. We can compile this code by using the SDK. To do this we need to open a command window. Doing this is described above. Then we can navigate to the tutorial folder.

```
cd c:\wamp\www\tutorial
```

Once in the correct folder we can execute the following statement:

```
sdk\bin\mxmlc --strict=true --file-specs tutorial.mxml
```

When we look inside the folder we will now see an new file called tutorial.swf. This is the flash file generated. We can open this file with a browser like internet explorer or Firefox and see our application.

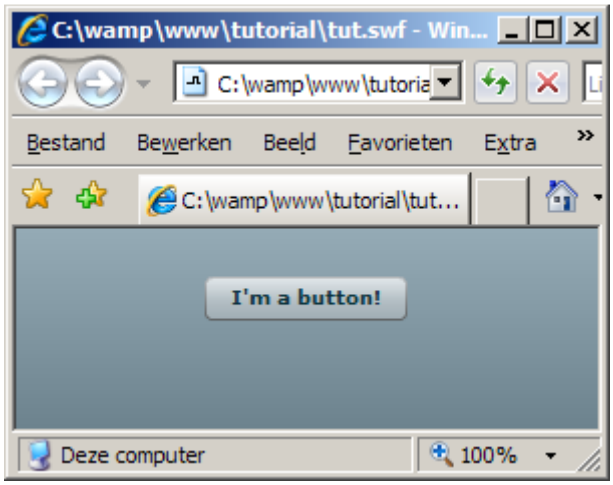


Figure 9.3.4: First Flex example, Hello World!

Information about compiling your first Flex project is also available on the Adobe website with some more detailed information (39). Once the default set up and installation is done we can continue with creating the website which is displaying/publishing the content which we can later change with the CMS we will be building.

1.4 Publishing website

Before we will start changing the generated website I will first walk through the files which have been generated.

```
Tutorial
|-- application              /* The folder where the logic will be     */
|   |-- Bootstrap.php        /* File which loads necessary files       */
|   |-- configs              /* Folder for configuration files         */
|   |   `-- application.ini  /* Configuration file for the app         */
|   |-- controllers          /* Folder which contains all controllers  */
|   |   |-- ErrorController.php
|   |   `-- IndexController.php
|   |-- models               /* Folder with the models for the DB      */
|   `-- views                /* Folder with the view scripts for GUI   */
|       |-- helpers          /* Scripts that are used often            */
|       `-- scripts          /* Default GUI scripts                    */
|           |-- error        /* Scripts for ErrorController            */
```

```
|           |   `-- error.phtml
|           `-- index         /* Scripts for IndexController           */
|               `-- index.phtml
|-- library                   /* Location of the Zend Framework        */
|-- public                    /* Files which are public (CSS/JS/Images) */
    `-- index.php             /* This file which is the startup        */
```

The information above might give some information but is still vague for new users of the Zend Framework. Therefore I will describe it more detailed. The entire application exists of three folders.

Application folder

The folder contains the logic of the website. Here we can describe the configuration of the website, the information about the database, how we want each page to look like and also which action will display which information.

The Zend Framework works with an URL rewriting system. This means that every request will be passed through the index.php file. This file then checks where the request should be redirected to. To give an example:

http://localhost/tutorial/public/

The URL above has no parameters and will only call the index.php file. Because there are no parameters this request will go to the default controller which is the IndexController and the IndexController will by default load the index.phtml script. Now another example:

http://localhost/tutorial/public/get/page

This URL will normally be redirected to the GetController. Within this controller it will look for the PageAction and display the correct script file which is in this case the page.html file from the page folder within the scripts folder.

If one of the controllers or actions is not found then the index.php will redirect to the ErrorController and display the correct error message.

Library folder

The library folder is the folder where all the default files of the Zend Framework are located. This folder shouldn't be changed at all. If you want to create your own Zend components/modules you might take a look inside this folder and get an idea about how all the modules and components are structured.

Public folder

The public folder is the folder to place the files which are allowed to be accessed via the web server. An good example of this would be the JavaScript files or style sheets. Also uploaded or normal images should be placed inside this folder.

1.4.1 Database

For storing the content of the website we need to set up a database. Within the database we need to store the content of the page, the title, the meta information and the URL of the image of the page. When converting this to SQL the statement for creating this table will look like this:

```
-- Database: `tutorial`
-- --------------------------------------------------------
-- Tabel structuur voor tabel `pages`

CREATE TABLE IF NOT EXISTS `pages` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  `content` text,
  `meta` text,
  `image` varchar(255) DEFAULT '',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

For working with this table inside our publishing website we need to insert some records. For testing purpose I will insert three records.

```
INSERT INTO  `pages` (
   `id` ,
   `title` ,
   `content` ,
   `meta` ,
   `image`
) VALUES (
   '',  'Home',  'homepage',  'home,general',  ''
), (
   '',  'Page 1',  'page 1',  'page,first',  ''
), (
   '',  'Page 2',  'page 2',  'page,second',  ''
)
```

The next thing we need to do is set up the Zend Framework to work with the database otherwise we cannot connect to the database and use the pages which we want to display. Within the application folder we find a folder configs. Inside this folder we can open the application.ini file with a text editor. This file is divided into sections, the first one is productions followed by staging, development and testing. We need to add the configuration of the database to the first section.

```
resources.db.adapter = "pdo_mysql"
resources.db.params.host = "localhost"
resources.db.params.username = "root"
resources.db.params.password = ""
resources.db.params.dbname = "tutorial"
resources.db.isDefaultTableAdapter = true
```

Once this is done, we need to create a model file which will be coupled to the database table. To do this we go to the folder models inside the application folder. Here we will create a folder called DbTable (This because there might be other models which are not from a database). Then we will create a file with the name Pages.php. The content will be like the following code:

```
<?php
	class Model_DbTable_Pages extends Zend_Db_Table_Abstract {
		protected $_name = 'pages';

		public function getPage($id){
			$id = (int)$id;
			$row = $this->fetchRow('id = ' . $id);
			if (!$row) {
```

```
                        throw new Exception("Count not find row $id");
                }
                return $row->toArray();
        }
}
```

Notice that the PHP closing tag (?>) is missing. This is done on purpose and required by the Zend Framework. When we have created the model for the Pages table the only thing we need to do is load the models when the application is called by the browser. This is done by the bootstrap file which is also inside the application folder. The bootstrap should look like the code below when finished.

```
<?php

class Bootstrap extends Zend_Application_Bootstrap_Bootstrap {
        protected function _initAutoload() {
                $moduleLoader = new Zend_Application_Module_Autoloader(array(
                        'namespace' => '',
                        'basePath' => APPLICATION_PATH));
                return $moduleLoader;
        }
}
```

Now let's if we can display all the titles of the pages. Within the IndexController we can load all the pages which are stored inside the database. We can do this by adding two lines into the IndexAction.

```
$pages = new Model_DbTable_Pages();
$this->view->pages = $pages->fetchAll();
```

In the second line we have fetched all the records from the database and set this information in a variable which we can load from inside the view script. Notice the $this->view->…. Now let's change the code of the script file (scripts/index/index.phtml) and replace the current content with the following:

```
<?php
        foreach($this->pages as $page) {
                echo $page->title . "<br />";
        }
```

When going to the website (http://localhost/tutorial/public) we will see a website which displays three very simple text lines which are the titles of the pages in our database. This means the interaction with the database has been established successfully.

1.4.2 Partial script files

The final thing which is very might come in handy when using the Zend Framework are partial script files. Inside the scripts folder we can create files. These files can then be rendered from any other script file. Which means that if we want to use a menu inside each different controller we can create one simple partial script file and include this in all the scripts which are related to the controller. For example the scripts/index/index.phtml.

We can do this inside the index.phtml file by adding the following line:

```
        echo $this->render("partial-script-file.phtml");
```

1.4.3 Complete layout

Since the creating of the layout is outside of the scope of this project I will give the complete source code of the website. This is available on the master project's website and is an archive of the entire Zend Project so far.

http://www.trilectica.org/Master-Project

The archive needs to be extracted to the location where the first part of the tutorial was located. The curious persons might take a look at the source code of the IndexController and the GetController. The GetController is where the pages are actually rendered. This also includes a simple part of SEO. What actually happens is that the links in the menu are built up with the name of the page. This way the description of the page is better and the ranking will be higher as when the page is called with the ID from the database. There is however one downside, the name of a page should be unique. One solutions to fix this is to add some extra content. For example add the date into the URL or add the ID of the page besides the name of the page.

Now the publishing website is done we can continue with the CMS with which we can later change the content of the publishing website.

1.5 Editing website (CMS)

The first thing that needs to be done within this website is setting up authorization for the CMS. To do this we can do two things, have authorization for one single user and put this in the source code or we can insert the users inside an database table which makes it possible to add users or if needed delete users from the system. Since this is the best solution we will start the editing website with this.

1.5.1 Authorization of the CMS

For storing the users of the CMS we need to set up a table in the database. The SQL for this table will look like this.

```
-- Database: `tutorial`
-- --------------------------------------------------------
-- Tabel structuur voor tabel `users`

CREATE TABLE IF NOT EXISTS `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  `email` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  `active` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

For working with this table inside our publishing website we need to insert a record.

```
INSERT INTO  `users` (
   `id` ,
   `name` ,
   `email` ,
   `password` ,
   `active`
) VALUES (
   '',  'Stefan',  'stefan@trilectica.nl',  'e8636ea013e682faf61f56ce1cb1ab5c',  '1'
)
```

Executing these two statements into the tutorial database results into a new table with a entry which is a user with the name 'Stefan' and the password 'geheim'. Now we can continue building with Flex and use this user to login into the system.

The next thing we need to do is set up the Amf service with which we should be able to communicate with from the Flex CMS. To do this we have to create an action inside the system where we can get some information from. For the authorization the best url would be:

http://localhost/tutorial/public/get/user/type/amf.

If we want this to work we also have to define the userAction inside the GetController. Lets walk through the code of this action.

```
…
…
public function userAction()
{

        /*
         * Check if the request has the parameter amf.
         * Because of this parameter we can handle the request with AMF
         * and return the correct results.
         */

        if ($this->getRequest()->getParam('type') == 'amf'){
                // AMF request.
                $this->_helper->viewRenderer->setNoRender();
                // Instantiatie a new server to handle the request.
                $server = new Zend_Amf_Server();
                // Say to the server where his services can be found.
                $server->addDirectory('../application/services/');
                // Start handling the requests of the server.
                echo($server->handle());
        } else {
                // Normal Request.
        }
}
…
…
```

As you might have noticed in the code above, there is a services directory inside the application folder. We also have to create this folder. Inside this folder we can place the services which the CMS can use. For the login we will create a UserService.php. This file can then check if a user is authorized to access the system or not.

```
<?php
        class UserService
        {
                // The function to call from Flex
                public function loginAction($user, $password)
                {
                        $users = new Model_DbTable_Users();
                        $user = $users->validateUser($user, md5($password));
                        if ($user)
                                return $user->toArray();
                        else
                                return array();
                }
        }
```

The service above shares a function which is the loginAction. This function can be accessed from Flex and the return value can also be used from withing Flex. Notice that we return the result as an array. This is required since Flex 3 cannot work with PHP objects. As you also might notice is that we create a new Model_DbTable_Users object. This is the last thing we need to do before we have finished setting up the Amf part inside the PHP code. The Model_DbTable_Users will be defined in the application/models/DbTable folder. The name of the file should be Users.php in order to work. The content of the file will be the following.

```
<?php
	class Model_DbTable_Users extends Zend_Db_Table_Abstract {
		protected $_name = 'users';

		public function validateUser($username, $password){
			$row = $this->fetchRow('name = "' . $username . '" and password = "'.
					$password .'" and active = 1');
			if (!$row) {
				return;
			}
			return $row;
		}
	}
```

This function will check the users table if there is a user with the corresponding name and password. Also will it check if the user is active and therefore can login to the CMS. If this is the case it will return the user otherwise it will return null which means the user is not authorized.

The next step in the authorization process it to build a simple form within Flex and call the Amf server with the corresponding function and see if we can login to the system. I will give the code of the tutorial.mxml file and walk through the code while explaining the code.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
	xmlns:mx="http://www.adobe.com/2006/mxml"
	horizontalAlign="center" verticalAlign="center">
	<mx:Script>
		<![CDATA[
			import mx.controls.Alert;	// import for alerting information
			import mx.rpc.events.*;		// import for the result of Amf request.


			// Function to handle the click on the login button.
			private function validateUser():void{
				// Send the remote object to the server and
				// set the variables (username and password)
				zendRemoteObject.loginAction(username.text, password.text);
			}

			// Function to handle the result of the login request.
			private function handleLoginAction(event:ResultEvent):void{
				// Check if we have a user back from the request
				if (!event.result.name) {
					// No user back. Display message.
					loginError.text = "Invalid login, try again.";
				} else {
					// User back, alert the username.
					loginError.text = "";
					Alert.show("Logged in as: " + event.result.name);
				}
			}
		]]>
	</mx:Script>
```

```
        // The object which handles the request to the server.
        <mx:RemoteObject      id="zendRemoteObject" destination="zend"
                              // The name of the service.
                              source="UserService"
                              // Where will te request go to?
                              endpoint="http://localhost/tutorial/public/get/user/type/amf"
                              // What to do when the result comes in.
                              result="handleLoginAction(event);"/>

        // The panel which contains the two input fields and the submit button.
        <mx:Panel      id="loginForm" width="300" height="175" horizontalCenter="0"
                       verticalCenter="0" title="Login to CMS:" paddingTop="10"
                       paddingLeft="10">
                <mx:HBox>
                       <mx:Label text="Username:" width="100" />
                       <mx:TextInput id="username" width="100" />
                </mx:HBox>
                <mx:HBox>
                       <mx:Label text="Password:" width="100" />
                       <mx:TextInput id="password" width="100" displayAsPassword="true" />
                </mx:HBox>
                <mx:HBox>
                       <mx:Label text="" width="100" />
                       <mx:Button     id="loginButon" width="100" label="Login"
                                      click="validateUser();" />
                </mx:HBox>
                <mx:HBox>
                       <mx:Label id="loginError" text="" width="200" color="red"/>
                </mx:HBox>
        </mx:Panel>
</mx:Application>
```

When looking at this code it all looks very straightforward. It might take some time before really knowing what actually happens. So what happens? When a user clicks on the login button the validateUser function is called. This function submits the remoteobject to the server with the filled in username and password. The application then wait for the result. Once the result has been send back to the application the handleLoginAction has been called with the result. The parameter which has been passed to this function is the result parameter. This parameter contains very much information about the actual request and result. The result value is part of the parameter and can be found as the 'event.result'. Since we send back the user object as an array, we can get the information of the user by typing for example: event.result.name or event.result.email.

Once we know we have a name, we know that we have found a user which meets the username and password. Knowing this we are logged in to the CMS and we can continue to editing the pages of the website.

Note that when we make any changes to the mxml code we do need to recompile the Flex Project with the batch file (tutorial.bat). If we don't do this the compiled swf file will not change.

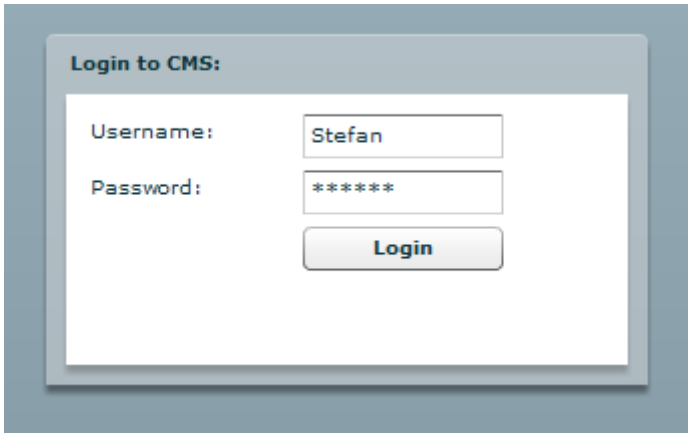


Figure 9.5.1.1: Flex CMS login screen.

1.5.2 Getting the pages from the database

Since we now know how to get information from Zend and know that the user is authorized we can continue to get the pages from the database and display them in a list. Therefore we need to alter the function which handles the login action.

```
…
…
        <mx:Script>
                <![CDATA[
                        import mx.controls.Alert;
                        import mx.rpc.events.*;

                        // array with the information of the user (name, email, etc…)
                        private var _user:Array;

                        // Getter for the user array.
                        public function get user():Array {
                                return _user;
                        }

                        // Setter for the user array.
                        public function set user(user:Array):void{
                                _user = user;
                        }

                        // Unchanged validateUser function
                        private function validateUser():void{
                                zendRemoteObject.loginAction(username.text, password.text);
                        }

                        // The changed handle login action
                        private function handleLoginAction(event:ResultEvent):void{
                                // set the user variable with the requests result.
                                user = event.result as Array;

                                // Check on the array instead of the result.
                                if (!user.name) {
                                        loginError.text = "Invalid login, try again.";
                                } else {
                                        loginError.text = "";
                                        // if logged in, remove the login box.
                                        removeChild(loginForm);
                                        // The pgEditor will be a panel where the pages can
                                        // be edited. Here we set the status with the name and
                                        // email address of the user.
                                        pgEditor.status = "Logged in as: " + user.name + "  (" +
                                                          user.email + ")";
```

```
                                        // We set the visibility of the editor to true.
                                        pgEditor.visible = true;
                                }
                        }
                ]]>
        </mx:Script>
…
…
```

What this does is remove the login form from the application and then shows the page editor panel. As we haven't created this panel before this should be instantiated. This will be done as a custom component. This means that if we want it can be re-used on other places without copying the code into multiple places.

Custom components is an important part of the Flex structure and therefore I will also explain how this works. Since Flex uses namespaces for separating the different parts from a system we will do this the same time.

Inside the tutorial folder we will create a folder called components. In there we will create a file called "PageEditor.mxml". This will be the component which contains the editor and the tree which will load the pages from the Zend framework and database.

To do this with steps I will now give the code of the first part of the page editor component. The comments inside the code will describe what happens.

```
<mx:Panel
        // The namespace where the definition of the objects can be found.
        xmlns:mx="http://www.adobe.com/2006/mxml" width="1000" height="600"
        // The title of the panel is set.
        title="VU :: Content Management System"
        // What to do when the panel gets visible?
        show="getPages();">
        <mx:Script>
                <![CDATA[
                        import mx.controls.Alert;
                        import mx.rpc.events.*;

                        // Variable which will later contain all the pages loaded from Zend.
                        // Bindable means it might be set from elsewhere.
                        [Bindable]
                        private var pagesArray:Array = new Array();

                        // Load all the pages when the panel gets visible.
                        private function getPages():void{
                                // Send the request.
                                zendRemoteObject.getAllPagesAction();
                        }

                        // Handle the requests result.
                        private function handlePagesAction(event:ResultEvent):void{
                                // Set the pages array with the result of the request.
                                pagesArray = event.result as Array;
                        }

                        // The tree must know which part of the arrays should be the label
                        // of the tree.
                        private function tree_labelFunc(item:Array):String {
                                // Set the label.
                                return item.title;
                        }
```

```
		]]>
	</mx:Script>
	// The remoteobject
	<mx:RemoteObject id="zendRemoteObject" destination="zend"
		// The service which needs to be called.
		source="PageService"
		// Where can the service be found. (notice the page action.
		endpoint=http://localhost/tutorial/public/get/page/type/amf
		// what do we do with the result?
		result="handlePagesAction(event);"/>

	<mx:HBox	paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom="10"
			height="100%">
		<mx:VBox width="200" height="100%">
			<mx:Label text="Pages inside system:" />
			// The tree instance where the pages will be places in once loaded.
			<mx:Tree id="tree"
				// Set the dataprovider as the pagesArray.
				dataProvider="{pagesArray}"
				// Define how to handle the labeling of the items.
				labelFunction="tree_labelFunc"
				width="175" height="100%"/>

		</mx:VBox>
		<mx:VBox>
			<mx:Label text="Select a page to edit it." />
		</mx:VBox>
	</mx:HBox>
</mx:Panel>
```

Since we now have a panel which should be able to load information from Zend we need to instantiate this panel and display it inside the swf. Therefore we first need to set a new namespace inside the tutorial.mxml file. We can do this by updating the application tag like the following:

```
<mx:Application
	xmlns:mx="http://www.adobe.com/2006/mxml"

	// The namespace for the components folder.
	xmlns:local="components.*"
	horizontalAlign="center" verticalAlign="center">
	…
	…
```

And at the bottom we need to instantiate the object like this:

```
	…
	…
	<local:PageEditor id="pgEditor" visible="false" />
</mx:Application>
```

If we run the batch file and refresh the browser we still see the login form. If we login now we see that the login panel is being removed and that we see a new panel which contains a tree on the left side. Inside this tree we find the three pages which we inserted into the database.

Figure 9.5.2.1, Example of page editor with Flex Tree structure.

The next step in this tutorial is adding the functionality to change and save the content of the pages which we have stored in the database. This includes adding an image to the page. The content can be changed using an WYSIWYG editor.

1.5.3 Changing the content of a page in the system

Since we already have the tree inside our editor the next step is adding an handler to the tree for when we are clicking on one of the items (an page). This can be done by changing the tree tag to the following piece of code.

```
…
…
<mx:Tree id="tree"
        dataProvider="{pagesArray}"
        labelFunction="tree_labelFunc"
        width="175" height="100%"
        // The handler for the page click.
        click="displayPage(event)"/>
…
…
```

Once the handler is added we need to defile the displayPage function inside the script tag. This function should then display the contents of the page into an WYSIWYG editor. Let's assume that the name of the WYSIWYG editor will be 'editor'.

```
…
…
private function displayPage(event:MouseEvent):void{
        // Here we set the htmlText of the editor. The content is retrieved
        // from the pagesArray. We know which item we should get from the array
        // by the selectedIndex of the tree. Not the fromHTML function.
        // This function is needed because Flex works with another standard for
        // HTML as most browsers. This function will be described below.
        editor.htmlText = fromHTML(pagesArray[tree.selectedIndex].content);
        // Set the title of the editor to the pages title.
        editor.title = pagesArray[tree.selectedIndex].title;
}
…
…
```

I will give the functions to convert the Flex html to normal valid html and back. These functions and more information about them can be found here: (http://thanksmister.com/?p=17). They are also given below.

```
<!--

	flex conversion functions from: http://thanksmister.com/?p=17

-->


		private function toHTML(str:String):String
		{
			var pattern:RegExp;
			pattern = /<TEXTFORMAT.*?>/g;
			str = str.replace(pattern, "");
			pattern = /<\/TEXTFORMAT.*?>/g;
			str = str.replace(pattern, "");
			pattern = /<P ALIGN="LEFT">/g;
			str = str.replace(pattern, "<p style=\"text-align:left\">");
			pattern = /<P ALIGN="RIGHT">/g;
			str = str.replace(pattern, "<p style=\"text-align:right\">");
			pattern = /<P ALIGN="JUSTIFY">/g;
			str = str.replace(pattern, "<p style=\"text-align:justify\">");
			pattern = /<\/P>/g;
			str = str.replace(pattern, "</p>");
			pattern = /<FONT (.*?)>/g;
			str = str.replace(pattern, "<span style=\"$1\">");
			pattern = /COLOR=\"(.*?)\"/g;
			str = str.replace(pattern, "color:$1;");
			pattern = /SIZE=\"(.*?)\"/g;
			str = str.replace(pattern, "font-size:$1px;");
			pattern = /FACE=\"(.*?)\"/g;
			str = str.replace(pattern, "font-family:$1;");
			pattern = /ALIGN=\"(.*?)\"/g;
			str = str.replace(pattern, "text-align:$1;");
			pattern = /LETTERSPACING=\".*?\"/g;
			str = str.replace(pattern, "");
			pattern = /KERNING=\".*?\"/g;
			str = str.replace(pattern, "");
			pattern = /<\/FONT.*?>/g;
			str = str.replace(pattern, "</span>");
			pattern= /<\/LI><LI>/g;
			str = str.replace(pattern, "</li><li>");
			pattern= /<\/LI>/g;
			str = str.replace(pattern, "</li></ul>");
			pattern= /<LI>/g;
			str = str.replace(pattern, "<ul><li>");
			pattern = /<I>/g;
			str = str.replace(pattern, "<em>");
			pattern = /<\/I>/g;
			str = str.replace(pattern, "</em>");
			pattern = /<B>/g;
			str = str.replace(pattern, "<strong>");
			pattern = /<\/B>/g;
			str = str.replace(pattern, "</strong>");
			pattern = /<U>/g;
			str = str.replace(pattern, "<u>");
			pattern = /<\/U>/g;
			str = str.replace(pattern, "</u>");


			return str;
		}
```

```
private function fromHTML(str:String):String
{
	var pattern:RegExp;
	pattern = /<p style="text-align:left">/g;
	str = str.replace(pattern, "<P ALIGN=\"LEFT\">");
	pattern = /<p style="text-align:right">/g;
	str = str.replace(pattern, "<P ALIGN=\"RIGHT\">");
	pattern = /<p style="text-align:justify">/g;
	str = str.replace(pattern, "<P ALIGN=\"JUSTIFY\">");
	pattern = /<\/p>/g;
	str = str.replace(pattern, "</P>");
	pattern = /<span style=\"(.*?)\">/g;
	str = str.replace(pattern, "<FONT $1>");
	pattern = /color:(.*?);/g;
	str = str.replace(pattern, "COLOR=\"$1\" ");
	pattern = /font-size:(.*?)px;/g;
	str = str.replace(pattern, "SIZE=\"$1\" ");
	pattern = /font-family:(.*?);/g;
	str = str.replace(pattern, "FACE=\"$1\" ");
	pattern = /text-align:(.*?);/g;
	str = str.replace(pattern, "ALIGN=\"$1\" ");
	pattern = /<\/span.*?>/g;
	str = str.replace(pattern, "</FONT>");
	pattern= /<\/li><li>/g;
	str = str.replace(pattern, "</LI><LI>");
	pattern= /<\/li><\/ul>/g;
	str = str.replace(pattern, "</LI>");
	pattern= /<ul><li>/g;
	str = str.replace(pattern, "<LI>");
	pattern = /<em>/g;
	str = str.replace(pattern, "<I>");
	pattern = /<\/em>/g;
	str = str.replace(pattern, "</I>");
	pattern = /<strong>/g;
	str = str.replace(pattern, "<B>");
	pattern = /<\/strong>/g;
	str = str.replace(pattern, "</B>");
	pattern = /<u>/g;
	str = str.replace(pattern, "<U>");
	pattern = /<\/u>/g;
	str = str.replace(pattern, "</U>");
	pattern = /  /g;
	str = str.replace(pattern, " ");


	return str;
}
```

```
<!--

	end conversion functions.

-->
```

And of course we need an instance of the editor on the page. This is an default Flex object which we can easily use on the website. The last VBox from the PanelEditor.mxml has to be updated with the following code.

```
…
…
<mx:VBox paddingLeft="10" height="100%" width="100%">
        <mx:HBox width="100%">
                // default header.
                <mx:Label text="Select a page to edit it." width="75%"/>
                // already inserted button which should later handle the save action.
                <mx:Button width="25%" label="Save Page" />
        </mx:HBox>
        // The RichTextEditor which will later contain the content of the pages.
        <mx:RichTextEditor    id="editor" text=""
                              width="100%" height="100%" />
</mx:VBox>
…
…
```

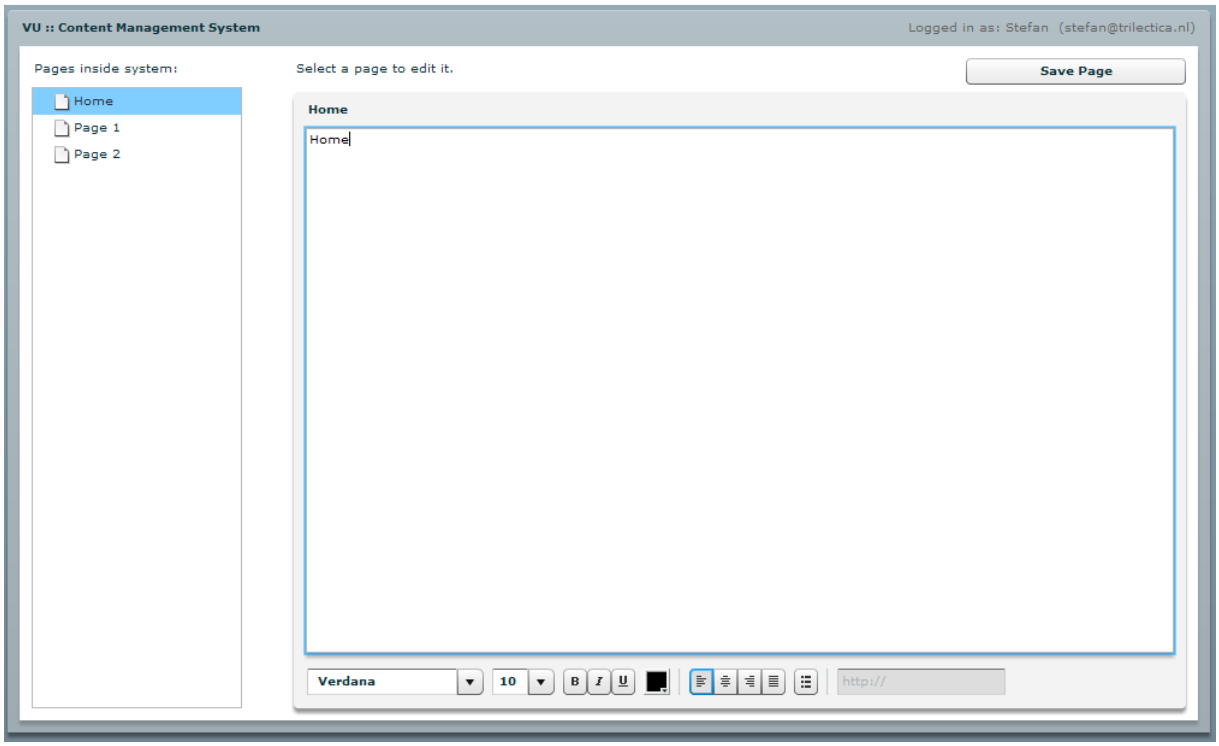


Figure 9.5.3.1: Flex CMS editor with WYSIWYG.

1.5.4 Changing the images of the page.

For each page we want to be able to insert, change and delete an image. How will we do this? Next to the editor we will show the image. When clicking on the image a popup will be shown where a different source for the image can be inserted. Let's continue to the source code for this.

```
<mx:VBox paddingLeft="10" height="100%" width="100%">
        <mx:HBox width="100%">
                <mx:Label text="Select a page to edit it." width="75%"/>
                <mx:Button width="25%" label="Save Page" />
        </mx:HBox>
        <mx:HBox width="100%" height="100%">
                <mx:RichTextEditor    id="editor" text=""
                                      width="75%" height="100%" />
                <mx:VBox width="25%" height="100%">
                        <mx:Image      id="pageImage" width="75%"
```

```
                                    minHeight="50"
                                    toolTip="Click to change the image."
                                    click="changeImage()"/>
                        <mx:Label text="Click image to change." width="75%"/>
                </mx:VBox>
        </mx:HBox>
</mx:VBox>
```

This is how the last VBox will look like. This shows the image of the page (if there is one). And when we click the image or “noImage” picture we can select an new image. Let’s see the changeImage function which is called when we click the image.

```
// Imports needed for the functionality.
import mx.managers.PopUpManager;
import mx.events.CloseEvent;

// A variable which makes it possible to close the popup in another function.
private var win:ChangeImage;

// Changing the image.
private function changeImage():void{
        // Create a new window which will be used as a popup.
        win = PopUpManager.createPopUp(this.parent, ChangeImage, true) as ChangeImage;
        // Add an listener for when we close the popup.
        // This makes it possible to change the images url.
        win.addEventListener("CHANGE_IMAGE", closePopUpCloseEvent);
        // Display the popup in the center of the screen.
        PopUpManager.centerPopUp(win);
}

// Function which is called when we dispatch the CHANGE_IMAGE event.
private function closePopUpCloseEvent(evt:Event):void{
        // Set the new URL from the windows newImageUrl text input field.
        pagesArray[tree.selectedIndex].image = win.newImageUrl.text;
        // Set the new source inside the current screen.
        pageImage.source = win.newImageUrl.text;
        // Remove the popup from the screen.
        PopUpManager.removePopUp(win);
}
```

However, having this image here also requires the displayPage function to be altered. This because we need to set the souce of the image to the correct image. If no image is set we will display a no photo image.

```
private function displayPage(event:MouseEvent):void{
        // Set the HTML from the array.
        editor.htmlText = fromHTML(pagesArray[tree.selectedIndex].content);
        // Set the title of the page.
        editor.title = pagesArray[tree.selectedIndex].title;

        // Check the source of the image.
        if (pagesArray[tree.selectedIndex].image != "")     {
                // There is an image, so set it.
                pageImage.source = pagesArray[tree.selectedIndex].image;
        } else {
                // No image, so display nophoto.
                pageImage.source = "nophoto.png";
        }
}
```

Another thing needed is the actual ChangeImage component. Here is how that component looks like.

```
<mx:TitleWindow
	xmlns:mx=http://www.adobe.com/2006/mxml
	layout="vertical"
	title="Test Popup"
	// Show the close button, for cancelling the current action.
	showCloseButton="true"
	width="400"
	height="150"
	// What to do when the window closes.
	close="titleWindow_close(event);"
>

	<mx:Script>
		<![CDATA[
			// The imports which are needed for the window.
			import mx.controls.Alert;
			import mx.events.CloseEvent;
			import mx.managers.PopUpManager;
			import mx.events.CloseEvent;

			// The function which is called when closing the window.
			private function titleWindow_close(evt:CloseEvent):void {
				// Remove the popup from the screen.
				PopUpManager.removePopUp(this);
			}

			// The function which is called when pressing the update button.
			private function throwEvent():void {
				// Instantiate a new event.
				var e:Event = new Event("CHANGE_IMAGE");
				// Dispatch the event to close the window.
				dispatchEvent(e);
			}
		]]>
	</mx:Script>
	<mx:VBox>
		<mx:Text text="Insert the URL of the image:"></mx:Text>
		<mx:HBox>
			<mx:Label text="URL:" width="100" />
			<mx:TextInput id="newImageUrl" width="100" />
	</mx:HBox>
		<mx:HBox>
			<mx:Label width="100" />
			<mx:Button	id="saveButon" width="100" label="Update"
					click="throwEvent()" />
		</mx:HBox>
	</mx:VBox>
</mx:TitleWindow>
```

1.5.5 Saving the page

If we want to add the functionality to save an page we need to add a new remote object.

```
<mx:RemoteObject	id="zendRemoteObject2" destination="zend" source="PageService"
			endpoint="http://cms.localhost/public/get/page/type/amf"
			result="handlePagesAction2(event);"/>
```

And here is the function which handles the result from the request. We only need to see if the result is true, otherwise something went wrong during the request.

```
private function handlePagesAction2(event:ResultEvent):void{
	if (event.result == true){
		Alert.show("Page saved successfully.");
	} else {
		Alert.show("Woops, something went wrong, please try again.");
```

```
	}
}
```

And of course the function which actually does the request. We update the array with the correct data first and then process the remote object to send the request.

```
private function savePage():void{
	// Set the content inside the array.
	pagesArray[tree.selectedIndex].content = editor.htmlText;
	// Send the remote object.
	zendRemoteObject2.savePage(   pagesArray[tree.selectedIndex].id,
				      toHTML(editor.htmlText),
				      pagesArray[tree.selectedIndex].meta,
				      pagesArray[tree.selectedIndex].image
			   );
}
```

As you might notice here we need to have an savePage function inside the PageService file. Let's take a look at the function which handles the saving of a page.

```
<?php

class PageService

{

	…
	…
	public function savePage($id, $content, $meta, $image){
		$data = array(
			'id'	=> $id,
			'content'	=> $content,
			'meta' => $meta,
			'image' => $image,
		);

		$page = new Model_DbTable_Pages();
		$page->save($data);
		return true;
	}
}
```

And the save function inside the Model_DbTable_Pages class.

```
public function save($data){
		$id = $data['id'];
		unset($data['id']);
		$this->update($data, 'id='.$id);
}
```

This code in total should now allow us to alter pages. We can add an image, remove the image by setting the images source to an empty field and save the pages once were done with editing the content of the page.

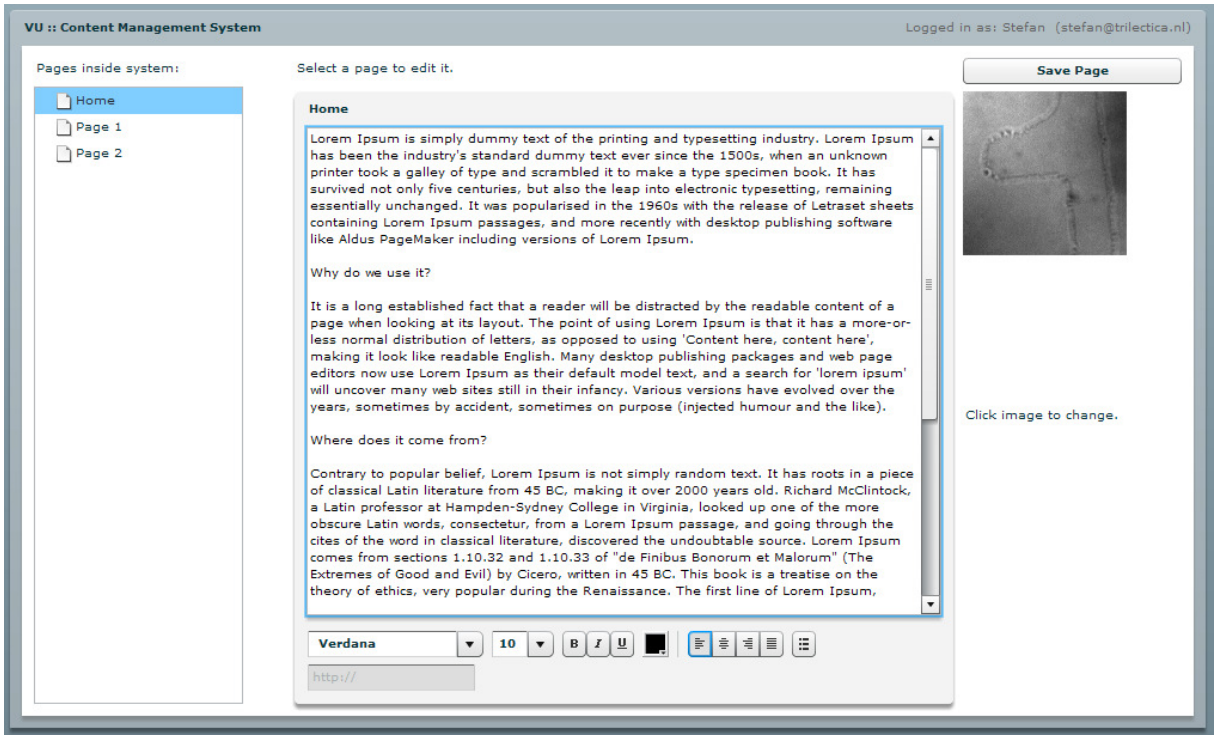


Figure 9.5.5.1: CMS editing page.

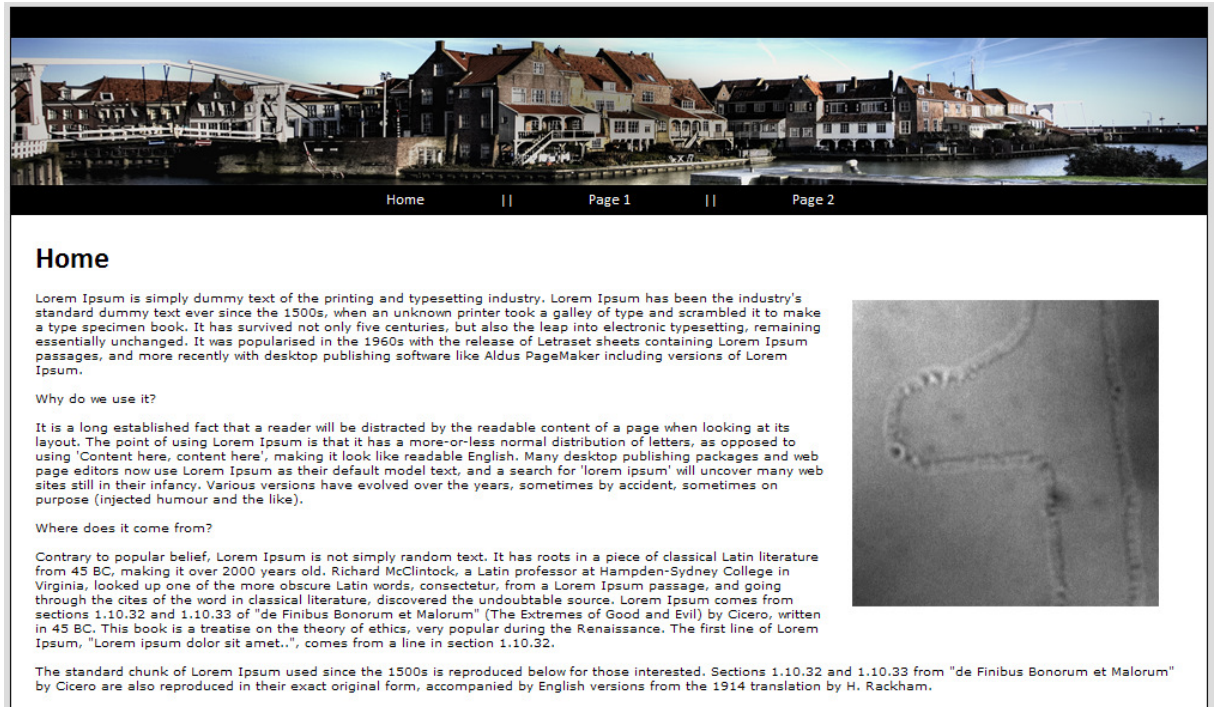


Figure 9.5.5.2: CMS Frontend.

1.6 Conclusion

We now have a fully working CMS which allows us to edit pages from the website and add or delete an image to each page. Some knowledge is required to fully understand what actually happens in the code. But once you get into working with Adobe Flex and the Zend Framework adding functionality to this CMS shouldn't be too difficult. Some things that might be added to this CMS might be adding or removing pages from the website. This can be done very easily by adding an RemoteObject which passes the correct page id to the delete it in the model or for changing the save method to check if the id is filled in. If there is no id set then insert instead a new record instead of updating the correct row attached to the given id.

This tutorial should give you a good idea about how Flex works in combination with Zend and what the possibilities of this combination are.