Kreus CMS & Fari MVC Framework Dissertation

Author
Radek Stepan
Candidate 45004

Project Supervisor
Dr Ian Wakeman

April 30, 2009

Statement of Originality

This report is submitted as a part requirement for the degree of Bachelor of Science at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

RADEK STEPAN

Acknowledgments

I would like to thank Dr Ian Wakeman for his guidance on this project and for letting me use agile development principles.

Legal Notice

Kreus CMS is released under the Open Source MIT License[1], which gives you the possibility to use it and modify it in all circumstances.

This license is GNU General Public License compatible, meaning that the GPL permits combination and redistribution with software that uses the MIT License.

[1] For more information about MIT License visit `http://www.opensource.org/licenses/mit-license.php`.

Summary

This project involves the development of a content management system called KREUS[2] based on an MVC framework called FARI[3].

The CMS uses features tailored to its end users, beginners; bridging the gap between not using a CMS at all and using a CMS like Joomla[4].

The developed system allows for basic creation, editing and backup of Web site content while allowing for more advanced features like file handling, or seamless contact form and sitemap inclusion.

End users were involved at key stages of the development while the actual development and project management followed agile and rapid prototyping principles.

Users had their say at the end of the development process using the system for a month on live Web sites, while having a quick way of asking for help throughout the testing period. The evaluation revealed that non tech-savvy users need to have the concept of CMS and separation of concerns explained while more advanced users were lacking more control over the system.

Path for further development and work is detailed at the end of the work.

[2]Kreus means 'icy cold' in Greek; the system was developed while the weather was icy cold.
[3]Fari is a Greek sounding word without a meaning.
[4]For more information about Joomla CMS visit `http://www.joomla.org/`.

Contents

Preface 11

Professional Considerations 11

Conventions 11

Report Structure 12

I The Idea of a CMS 14

1 Introduction 14

2 Methodology 14

3 Requirements 15
- 3.1 Functional Requirements 16
- 3.2 Primary Objectives 16
  - 3.2.1 User Interface 16
  - 3.2.2 Content 16
  - 3.2.3 Security 17
- 3.3 Secondary Objectives 17
  - 3.3.1 User Interface 17
  - 3.3.2 Content 17
  - 3.3.3 Security 17
- 3.4 Non-functional Requirements (Qualities) 18

II User Interface 19

4 Interface First 19

5 Three State Solution 19

III MVC Framework 20

6 Separation of Concerns 20

7 MVC Architecture 21
- 7.1 The Problem 21
- 7.2 Framework Solution 21
  - 7.2.1 URL Rewriting 21
- 7.3 The Problem 22
- 7.4 Framework Solution 22
  - 7.4.1 Auto-loading of Components 22
- 7.5 The Problem 22
- 7.6 Framework Solution 22
  - 7.6.1 Registry Component for Object Sharing 22
- 7.7 The Problem 23
- 7.8 Framework Solution 23

7.8.1 Router Component Determining Page Requested 23
7.9 The Problem 24
7.10 Framework Solution 24
7.10.1 Controller Component 24
7.11 The Problem 25
7.12 Framework Solution 25
7.12.1 Model, the Business Logic of the Application 25
7.12.2 View displaying Designed Templates 25
7.12.3 Extracting Variables for the Template 25

8 Writing Applications in Fari MVC 26

9 Discussion 27

IV CMS 28

10 CMS Architecture 28

11 Database and Data Objects 28
11.1 The Problem 28
11.2 Framework Solution 29
11.2.1 Database Structure 29
11.2.2 Handling Databases Easily with PDO 30
11.2.3 The Singleton Pattern 30
11.2.4 CRUD as a Database Abstraction Library 31
11.2.5 Discussion 31

12 Security 31
12.1 The Problem 31
12.2 Framework Solution 32
12.2.1 User Access Control 32
12.2.2 Form Token 32
12.2.3 Encryption of Data 33
12.2.4 Type Checking 33
12.2.5 Character Escaping 34
12.2.6 Securing Application Components 35
12.2.7 Securing File Access and Handling File Uploads 35
12.2.8 Discussion 36

13 Content Backup 36
13.1 The Problem 36
13.2 Framework Solution 36
13.2.1 XML 36
13.2.2 Discussion 38

14 Languages 38
14.1 The Problem 38
14.2 Framework Solution 38
14.2.1 Handling Multiple Languages in Code 38

15 Search Engine Optimization 39
15.1 The Problem 39
15.2 Framework Solution 39
15.2.1 Optimization 39
15.2.2 Keywords 39
15.2.3 Sitemap 40

16 User Input 40
16.1 The Problem 40
16.2 Framework Solution 41
16.2.1 Lightweight Markup Language 41
16.2.2 Rich Text Editor 41
16.2.3 Discussion 42

17 Site Content 42
17.1 The Problem 42
17.2 Framework Solution 43
17.2.1 Content Structure & Editing 43
17.2.2 Serving Pages 44
17.2.3 Discussion 45
17.2.4 Generating a Secure Contact Form 45
17.2.5 Sending Email 46
17.2.6 Usable Forms via Ajax 46

V User Evaluation 48

18 Introduction 48
18.1 Evaluation of Software Quality 48

19 Results 48
19.1 Discussion 50

VI Conclusions 51

20 Future of Kreus CMS 51

Appendixes 54

Credits 54

Glossary 55

References 59

Index 61

A System Evaluation 63
A.1 Questionnaire 63
A.2 Results 64
A.3 Comments 64

B XHTML Prototypes 66

C Kreus CMS Admin Interface 69

D Kreus CMS Browser Compatibility 72

E XSS 75

F Fari MVC Framework Screenshots 76

G Fari MVC Framework Q&A 77

H Project Management & Updates 78
H.1 Preparation & Setup 78
H.2 XHTML Prototypes 78
H.3 Model-View-Controller Framework 79
H.4 Content Management System Development 80
H.5 Further CMS Development 81

I Kreus CMS Source Code 82

List of Figures

1 *Content management* used at one renowned University. 14
2 Blank state of the Pages screen. 19
3 Decoupling of PHP code and HTML tags. 20
4 MVC Framework processing a request. 21
5 Listing of models within Kreus CMS. 28
6 Many-to-many relationship in a relational database. 29
7 Database structure of the *pages* table. 29
8 Site settings. 40
9 Editing a page content in Kreus CMS. 42
10 Admin interface in sNews CMS. 43
11 Editing a page in Kreus CMS. 44
12 Kreus CMS software quality assessment results. 48
13 Comparison of code and comments size for Frameworks and CMS. 51
14 System evaluation results. 64
15 First XHTML Prototype. 66
16 Second XHTML Prototype. 66
17 Third XHTML Prototype. 67
18 Fourth XHTML Prototype. 67
19 Final XHTML Prototype of all screens. 68
20 Kreus CMS login screen. 69
21 Kreus CMS pages screen. 69
22 Kreus CMS edit page screen. 70
23 Kreus CMS files screen. 70
24 Kreus CMS backup screen. 71
25 Kreus CMS settings screen. 71
26 Kreus CMS under Mozilla Firefox 3. 72
27 Kreus CMS under Opera. 72
28 Kreus CMS under Safari. 73
29 Kreus CMS under Chrome. 73
30 Kreus CMS under Internet Explorer 8. 74
31 Acunetix Web Vulnerability Scanner XSS scan result. 75
32 Successful XSS against Frog CMS. 75
33 Fari Framework handling an exception of a missing view. 76
34 Fari Framework handling an exception on a production server. 76
35 Preparation & Setup stage. 78
36 XHTML Prototyping stage. 78
37 MVC Framework development stage. 79
38 CMS development stage. 80
39 Further CMS development stage. 81

Preface

Professional Considerations

1. Kreus CMS is **open source** software, meaning wide public has access to its source code. In addition to this, the system implements a possibility to use a different system language by design. This is to allow the wide community to tailor the system to its needs. As an example the language of Hebrew or Arabic is written predominantly right-to-left and thus the user interface needs to be tailored for this purpose.

2. In addition to being open source, the source code is inline style **commented** to allow for peer reviews, collaboration and professional development of others.

3. The system's programming language of PHP is translated to plain HTML and CSS on the server and care has been taken to make sure it is semantically valid, thus able to be displayed on a variety of Internet browsers (Appendix D) that users wish or might have to use. Semantic validity is being checked by W3C. It is important that browsers viewing this CMS are compatible as per W3C requirements.

4. **HUMANS**, not code validators, use Web interfaces. User input is being checked in key stages of the development.

5. CMS poses no conflict of interest to University of Sussex authorities and will be completed within agreed time scales. The work committed to will be undertaken.

6. The level of work is appropriate to my level of skill and the technology used (PHP, MySQL, JavaScript, CSS, XHTML) is within my competence and I accept full responsibility for this project, not only that, I am proud of it.

Conventions

In this report, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words, file names and links in the text are shown as follows: "...we use magic methods *__get* and *__set* to handle any undefined variables by using the following code within the registry."

A block of code will be set as follows:

```
$registry = new Fari_Registry();
$registry->db = db::connect(Fari_Settings::get_db());
```

Important words are introduced in a bold-type font while IMPORTANT NAMES are in uppercase.

A Glossary is provided towards the end for easy look up of terms.

What This Report Covers

The Idea of a CMS

Section 1: This first chapter explains **why** there is a need for a user friendly content management system tailored to its users, beginners.

Section 2: This chapter details the **methodology** and principles used while working on the project.

Section 3: This chapter talks about what parts a CMS **requires**.

User Interface

Section 4: This chapter talks about the need for a usable **interface** and how to go about designing it.

Section 5: This chapter explains the methodology used when creating the **interaction** between the user and the system.

Model View Controller Framework

The following sections are formed in a problem - solution manner.

Section 6: This chapter takes us through the idea of using a **framework** as a basis for a Web application.

Section 7: This chapter explains what parts a full **MVC** framework contains and how these components fit together to support the application. The chapter is exhaustive in showing applied solutions to common problems and patterns used.

Section 8: This chapter shows us an **example** of how to develop an application based on Fari MVC Framework and thus also shows an interesting way of authorizing users.

Section 9: This chapter **discusses** what has been learned taking a broad perspective and describing a mental model.

Content Management System

The following sections are formed in a problem - solution manner.

Section 10: This chapter shows **models** used when building Kreus CMS.

Section 11: This chapter talks about **database** & data objects (PDO, Singleton and CRUD) and how they fit within the system.

Section 12: This chapter discusses **security** issues (UAC, tokens, filtering and escaping, XSS) pertaining to the project and how they are solved.

Section 13: This chapter explains the **import/export** facility provided to the end user via XML.

Section 14: This chapter details **languages** handling within Kreus CMS and the model used.

Section 15: This chapter discusses seamless **search engine optimization** (keywords and sitemap) of a Web site running on Kreus CMS.

Section 16: This chapter takes us through the processing of **user input** and facilities (LML and RTE) provided.

Section 17: This chapter explains the bread and butter of any CMS, the separation and handling of **site content**.

User Evaluation

Section 18: This chapter details the **evaluation** of Kreus CMS software quality by end users.

Section 19: This chapter talks about the **results** of system evaluation.

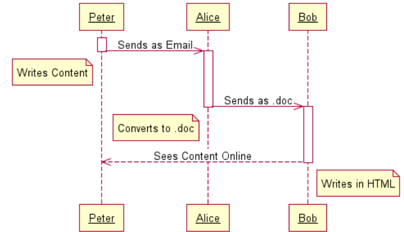
Conclusions

Section 20: This chapter talks about the **future direction** Kreus CMS and Fari Framework might take.

Part I
The Idea of a CMS

1 Introduction

Figure 1: *Content management* used at one renowned University.



Content management systems (CMS) are tools that help people to create and manage content in an efficient manner. Content could encompass articles published via blogs, items for sale in e-commerce setting, work collaborated on in a company, images in image galleries or pages of a company's Web presence. The managed content is wide and varied and so are solutions tailored to manage it[5,6].

CMS usually allows multiple users to access and manage content, sometimes users have different levels of access (privileges) and a work flow is in place where one user (author) submits a content and another user (editor, admin) needs to allow it so that it is published. These systems also allow a further separation of content from its layout.

This report describes how a niche has been found for a CMS managing Web presentations, its objectives and progress.

2 Methodology

This project uses **AGILE SOFTWARE DEVELOPMENT**[7] (rapid application development) principles at its core. This means fewer preferences in the system, fewer abstractions, less headache, fewer promises. At the beginning a very lean set of features is agreed upon and then an initial low-level prototype is built in XHTML and CSS. This allows users to quickly test the system and give insight at the beginning of the project which is cost (time) effective. Then a basic structure (called Fari MVC Framework) is rapidly developed through multiple iterations. The goal is to stay lean now.

[5]For live demo versions of major CMS visit `http://www.opensourcecms.com`.
[6]For a comparison of various CMS visit `http://www.cmsmatrix.org`.
[7]For more information on applied agile web development visit `http://gettingreal.37signals.com`.

What follows is a process of connecting the templates (XHTML, CSS) with the framework via continuous iterations. The MVC structure allows to test parts of the system before another part is complete and allows further and further iterations while the CMS can be test-driven. Practically, a function is coded in PHP and then tested in the real life. If the function works, it is then continually fine tuned and parts of it (or whole) reused.

The reason for following these principles is to have a piece of software very **early on** and a polished application **on time**. Features can be added/removed throughout the process and without much effort rather than following a rigid plan. The focus is on the **interaction** with users and a good design behind the scenes. Success of the project is measured by **what works** rather than what 'could have been'.

3 Requirements

Small and medium businesses usually cannot afford a dedicated editor taking care of their company's presentation on the Web. From personal experience people go for either of two solutions: they hire a Web designer that will do the job or search for an application that will do the task. Going for the former, they might end up being told to use a WYSIWYG solution like Microsoft FrontPage™ or Microsoft Word™ and then submit this solution to the Web designer (Figure 1), this process adds layers of extra work that makes the Web presentation less competitive than if they go for the latter.

The latter has been already introduced, it is the CMS. There are plenty of tools to play with for technical people but as is still common, fewer for less tech savvy ones. Whenever someone asked for a simple CMS that would 'do the job', I could not recommend an appropriate solution. There are plenty of tools to edit the content manually and then, after a big leap, there are plenty of tools that will do significantly more than just edit content, so much more that the original purpose of the CMS and usability is hindered.

Agile Development

Agile development minimalist philosophy has been employed (Appendix H.1, Figure 35). Particularly that of 37SIGNALS[8], a company that stands behind products such as Basecamp or Backpack. Companies such as 37SIGNALS or WEBREAKSTUFF are successful because they involve users and leave the functionality to a bare minimum. To give an example, Basecamp serves as a project collaboration tool and unlike in their competitors' products, you cannot find labels labelling messages as either important or less important throughout the system. 37SIGNALS say that instead of adding a feature called 'labels' they leave it up to the users to figure out a way... and they did, by simply adding a text label in front of a message [37sig06] like this:

```
[label] This is a message.
```

The aim of Kreus CMS is to get out of the way of people maintaining company Web presence and make this process as straightforward as possible as users do not want to 'use' the system but rather get the job done. This is the **vision** of this project.

[8] For more information visit `http://www.37signals.com`.

3.1 Functional Requirements

- User interface with a text editor that enables users to **create/edit** XHTML content of their website.
- System **generates** user friendly URLs, sitemaps , contact forms , links, SEO keywords.
- A way for users to **import/export** content.

3.2 Primary Objectives

Objective for this CMS is an editor of Web pages that is to be used even by people that are not very tech savvy. Focus will be on the **interaction** between the user and the application so that the application gets out of the way and does not cause clutter. This project is released under open source license for the wider community to work on and extend with extensive **documentation** alongside the source code[9].

A PHP framework based on MVC architecture is implemented and written from scratch and used as a basis for this and other projects.

Following are features and functionality that needed to be implemented that complement the user in achieving her goal, that is, simple publishing and managing of Web pages.

3.2.1 User Interface

- **Texy!/Textile enabled editor** for content creation and editing. Texy, a formatting tool, allows user to enter content using an easy to read syntax which is then converted into semantically valid XHTML. The editor will allow users to click on shortcut buttons for commonly used functions like setting the typeface bold, adding of images etc.
- Support for **multiple languages** that can be switched via settings page. Labels are variables and thus even a non-technical user can create translation of the system to their native language. This allows for wider user base and collaborative work on the system.
- Asynchronous JavaScript (**Ajax**) will be used when submitting/editing content, settings etc. (forms). Imagine a situation where you submit a form only to find out that you typed in an error and the content of the form has not been saved. Ajax combats this problem by not refreshing the page (redirect).

3.2.2 Content

- A page is split into main content and extra content (sidebar). This model should work in majority of small business websites while not cluttering the interface with categories, sub-categories, articles and pages all meaning the same.
- Ability to upload **files** to a public folder on the server. Framework will already allow for this model to happen, where application, framework and public files will be split on the server for a better security control.

[9] As of Kreus CMS 1.0.0 released Feb 13, 2009 full 52.66% lines of code are commented on in a consistent manner.

- **Sitemap** generated from pages for search engines. This is useful for business users.
- Pages can have **keywords** set for search engines, helping SEO and thus prominence of the website.
- User friendly URLs accessed via *mod_rewrite*. framework is at the heart of accessing the content.
- Generation of a content form with *mail()*. Contact page is a standard component of websites; it will help users by automatically generating one for them.
- **Backup** of content into XML or CSV so that transfer from one system/server to another can be made easy while protecting users' data.

3.2.3 Security

- Two levels of users, one editor and one admin with different levels of access. As the name suggests, an editor would only write and update content while administrator would have access to all functionality.
- Password encrypted on the server.
- Form token to make sure data are submitted from the system and not from a third-party server.
- All **user input filtered** based on preferences set in the model .

3.3 Secondary Objectives

3.3.1 User Interface

- Reorganize links in the menu of the system according to the popularity of their use.

3.3.2 Content

- RSS feed generated from news section/blog.
- Edit content link in actual pages.
- SMTP used instead of *mail()*.

3.3.3 Security

- Log of actions of all users saved in the database (to assign blame if something goes wrong).

3.4 Non-functional Requirements (Qualities)

- **Maintainability** of the code that is **documented** throughout and version **controlled** via Git.
- **Extensibility** via external modules and code, **open source** system.
- **Scalability** of the database and code according to best PHP OOP and SQL practices.
- **Usability** for target audience, small to medium business owners and staff. System has **consistent** interface.
- **Reliability** of code will be tested throughout, common **security** hacks tested.
- **Platform** selected as PHP with SQL. Database model can connect to MySQL, SQ Lite, Pg SQL database via PDO for PHP. The Web server needs to have *mod_rewrite* enabled. Forms in the CMS use Ajax where appropriate provision of MooTools. The reason for selecting PHP is that programmers of all skill levels can use it and thus this project, once released, can be extended by a wide variety of authors, while PHP being a language that is very mature.
- **Consistent** use of database models so that functions can be reused (see Platform).

Part II

User Interface

4 Interface First

Too many applications start with a programming-first mentality. That is a bad idea. Programming is the heaviest component of building an application [RapidDev96, p.350], meaning it is the most expensive and hardest to change. Instead, we should start by designing the interface first [37sig06, ch.9].

Developing and testing prototypes through an iterative design approach creates the most useful and usable Web site [Usability06, p.2], it creates something real so that everyone can see what the application looks like on the screen [37sig06, ch.6].

5 Three State Solution

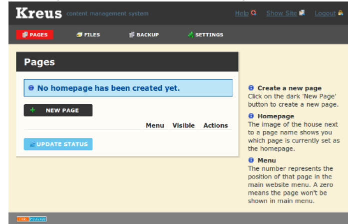
If the quality of the user's experience while interacting with a product suffers, then her impression of the product will suffer, no matter how technologically advanced and path-breaking the product might be [SoftArch09, ch.96].

For every screen we are creating, we need to consider three possible states:

- **Regular**: this is the state users will see when the application is filled with data; is up and running.
- **Blank**: this is the initial state users will see the application in. 'Lorem ipsum' looks good on the screen, but is that how the application looks like once we install it, fresh? An example is shown in Figure 2.
- **Error**: this is the state we need to account for when something goes wrong.

Once we have the initial prototype (Appendix B) tested with a few users (Appendix H.2, Figure 36), we can move on to the coding part. Of course, the development process does not stop us from further changes to the user interface if need be.

Figure 2: Blank state of the Pages screen.



Part III

MVC Framework

Model View Controller (MVC), first described by Trygve Reenskaug in 1979 [CodeIgnit07, p.28] at Xerox PARC, is a design pattern which simplifies application development by separating the application into three distinct parts (or components). The main reason it is used in application development is because of its clear separation of content and code and this allows developers and designers in larger teams to each work on their own distinct parts [ProPhp08, p.201]. It is a solution to tackle the problem of code reuse arising from practice, not theory [FactsEng03, ch.1].

The main reason MVC was picked as the core framework for this project was the promise of clean code creation and ease of updating.

6 Separation of Concerns

The first step was to understand how MVC works in the real life (Appendix H.3, Figure 37). There are many frameworks out there and a lot of them are written in PHP yet as a beginner my understanding of this paradigm was vague.

The whole concept was made more clear by a series of three blog posts [Nemetral08] that explain the pattern as it evolves from a traditional Web page script and thus as the PHP logic becomes separate from the presentation logic, thus in terms of MVC, how a model becomes separate from the view .

According to Nemetral[10] [Nemetral08]:

1. The first step is to move all PHP code that was up to now coupled with HTML tags, to the head of the page.
2. The second step is to move all HTML tags to a separate file and access it via a PHP *include*. So when a request is made it is directed to the PHP code (controller and model) and this code then requests the HTML tags or presentation (view) seen in Figure 3.

Figure 3: Decoupling of PHP code and HTML tags.

[10] This person wishes to stay anonymous.

7 MVC Architecture

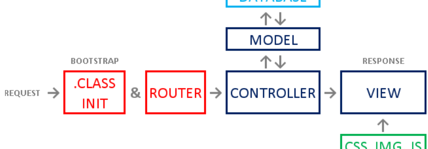
Full MVC makes use and separates all three parts of a typical application, thus:

MODEL is responsible for what is called a business logic; to write it more clearly it takes care of processing data and returning them to the controller for the view to display. As an example, it could be an object that will return a blog listing to us when we are viewing a blog application. Typically, a model will contain a database abstraction object [ProPhp08, p.212] so that all queries directed to the database are passed through it.

VIEW or templating is what Web designers take care of. This is the user interface people accessing the application see. Technologies exclusively found in the view are HTML, CSS and JavaScript [ProPhp08, p.201].

CONTROLLER changes the state of the model, loads models and provides dynamic data to the view [CodeIgnit07, p.28]. To put it simply, a controller glues the application together and when a request comes to the application, controller decides which action and thus model and view to load; example of which can be seen in Figure 4.

Figure 4: MVC Framework processing a request.



7.1 The Problem

The separation of code into distinct parts and how they connect together was then clear, but how does the application know which controller to load?

7.2 Framework Solution

7.2.1 URL Rewriting

Based on CodeIgniter framework, Fari framework makes use of *index.php* file which is a **MAIN POINT OF ACCESS** in the application where bootstrapping takes place. Whenever a request is made to the application the data are passed through this file. But how does the server do it?

Using **URL REWRITING** we can achieve this desired feature. In Apache, this is done by including an *.htaccess* file in the root directory of the application. Below is a part of such *.htaccess* file explaining how it takes all the parameters in the URL and passes them via *index.php* file as a parameter *route*.

```
# directs server to add the query string to the end of the url
    [QSA], last rule [L]
RewriteRule ^(.*)$ index.php?route=$1 [L,QSA]
```


7.3 The Problem

Now the application uses a central point of access so we can include the different parts of the MVC framework. As this can be seen and is thoroughly documented in the source code let's look at an interesting feature of PHP, **AUTO LOADING** of classes.

In a full application, it is expected to have many model objects. It is tedious (and was done in an initial version of the framework) to include each model manually one by one.

7.4 Framework Solution

7.4.1 Auto-loading of Components

Fortunately, PHP contains an auto loading mechanism which most MVC frameworks make use of [ProPhp08, p.207].

```
1 function __autoload($class_name) {
2    $f = BASEPATH . '/'. APP_DIR . '/models/' . $class_name .
        EXT;
3    try {
4       // check if class exists
5       if (!file_exists($f)) {
6          throw new Fari_Exception('Missing Model Class: ' . $f)
              ; // throw an exception
7       } else include($f); // include file
8    } catch (Fari_Exception $e) { $e->fire(); }
9 }
```

This function will try to include a class file of a model just before an **EXCEPTION** is thrown that a class has not been loaded. As can be seen in the example this advanced version also makes use of *Fari_Exception* class. PHP allows us to define our own exception handler and thus enables us to format the error messages one would see, in this example *Fari_Exception* handler throws a 'Missing model class: ' error if the class file has not been found.

7.5 The Problem

Now that we have included (or have a way of including) all of our classes that make up an MVC framework how do we link them together?

7.6 Framework Solution

7.6.1 Registry Component for Object Sharing

The first step was to leave a reference to components in global variables. It was then decided, based on the example of Zend framework, to use a registry object.

REGISTRY component is designed to allow us to create and share objects among the various framework classes. This approach is less likely to cause conflicts with other libraries that might be used in conjunction with the framework [ProPhp08, p.238].

Registry makes use of another two 'MAGIC METHODS' (the first one was *__autoload*) contained in PHP and that is *__get* and *__set*. This allows us to do the following:

```
$registry = new Fari_Registry();
$registry->db = db::connect($app_settings['db']);
```

In the example above, even though *db* has not been declared within the *registry* class, we can assign it values, and once assigned, we can retrieve those values. This is what is called UNDEFINED INSTANCE VARIABLES [OoPhp06, p.112] and we use magic methods *__get* and *__set* to handle any undefined variables by using the following code within the registry :

```
1  class Fari_Registry {
2     private $values = array();
3     function __set($name, $value) {$this->values[$name] = $value
         ;}
4     function __get($name) {
5        try {
6           // make sure the object exists, otherwise throw an
               Exception
7           if(!array_key_exists($name, $this->values)) {
8              throw new Fari_Exception('Object '.$name.' not
                  found in the Registry');
9           } else { return $this->values[$name]; }
10       } catch (Fari_Exception $e) { $e->fire(); }
11    }
12 }
```

7.7 The Problem

At this point the application can take in a request and load framework components, how does the application know which controller to load?

7.8 Framework Solution

7.8.1 Router Component Determining Page Requested

A **ROUTER** is a component working in conjunction with the *index.php* file as a single entry-point of the application [Nemetral08]. To put it simply, what we are doing is splitting the input route and figuring out which controller to load. In order to preserve a clean layout of the code, the router has been moved to a separate class file and works by processing a request:

```
GET http://localhost/kreus/admin/backup
```

We can see above the request coming from the Web browser. The *.htaccess* file attaches the */admin/index* part of the URL as a route variable and passes it to an *index.php* file. Index serving as a point of entry then loads a router that splits ('/') the route in *GET* variable into parts.


```
$route = explode('/', $request);
```

A process added in a later version cleans up the route and makes sure here and now that we don't process any XSS.

```
foreach ($route as &$part) {
   $part = preg_replace('/[^A-Z0-9:_\.-]/i', '', $part);
}
```

The first part of the array that is returned is the controller requested (*admin*), the second one is the action requested (*backup*).
An error check is then done to find out if we can load the controller.

```
!is_callable(array($controller, $this->action_requested))
```

If the controller is not found we can default to a controller of our choice. This is wonderful because we can load a default page (say *index*) for our pages listing in order to not throw the user a 404, Page Not Found Error.
If the controller is callable we include it and call the action contained.

7.9 The Problem

But what happens if the requested route does not contain an action? For example, when we are browsing a Web page `http://www.kreus-cms.com`, we unknowingly request an *index.html* or *index.php* file that resides on the server. We needed a way of doing the same thing using our framework.

7.10 Framework Solution

7.10.1 Controller Component

Thus when creating a controller object we need to specify a default and always included function *index*.
ABSTRACT FUNCTIONS are a powerful way that guarantees an implementation of particular functions. The declaration of an abstract *index* function ensures that we can not have a controller that does not have an *index* [OoPhp06, p.94].

```
abstract function index();
```

Then when we create an application controller we define it as extending our main *Fari_Controller*.

```
class Admin_controller extends Fari_controller {
   public function index() {;}
}
```

To get back to our exemplary route where we request the admin controller, the router locates the controller directory (*application/controllers*) and adds the name of the controller requested (*admin*) to it.

7.11 The Problem

Now that we have a controller loaded, we need to connect the model and view to it.

7.12 Framework Solution

7.12.1 Model, the Business Logic of the Application

As we have explored before, it is advisable to use a registry component instead of global variables. Yes, we are going to do the same here. The controller actually saves a registry for itself and thus we can save each data returned from the model into it and save it in another registry contained within the view .

```
$this->registry->view->message = files::delete($this->route[3])
    ;
```

The above line of code uses a registry of the controller to save a link to a view that itself has a registry value stored which is returned (in this case a string) object from a model called *files* and function *delete* with a parameter being a third value in the route requested.

7.12.2 View displaying Designed Templates

The simplest way to separate the programmers from the designers is to create two files for each URL. File 1 contains SQL statements and some procedural code that fills local variables or a data structure with information from the RDBMS. The last statement in File 1 is a call to a procedure that will fetch File 2, a **TEMPLATE** file that looks like standard XHTML with simple references to data prepared in File 1 [SoftEng06, p.147].

After we process the data from model(s) and save it for the view to use, we need to display the actual XHTML template .

```
$this->registry->view->display('admin/admin_login');
```

What we are doing on the line above is not saving a variable into the view registry but instead calling a *display* function with the parameter describing which view we want to load. This is brilliant. If we translate the line in to plain English we are saying: "take our Registry, give me the view and now display it".

As we would expect the view includes a *.tpl.php*[11] file inside the *admin* directory called *admin_login.tpl.php*. This is similar to a router including a controller.

7.12.3 Extracting Variables for the Template

How do we extract the values from the registry contained in the view now?

When setting these variables, we were creating a *key:value* array using magic method *__set* and now we need a way of extracting those values.

```
public function __set($index, $value) {
   $this->values[$index] = $value;
}
```

[11] As the view might have the same file name as a model (or a controller) we denote it with *.tpl*.

Based on the suggestion of ED ELLIOT [Elliot07] that has created a lite templating language, we can use a little known *extract* PHP function[12] that is functionally equivalent to the more often used:

```
foreach ($values as $index => $value) {
   $$index = $value;
}
```

As we are getting into the field of templating languages it needs to be mentioned that Fari framework contained a simple engine at the beginning of the development. The biggest problem and challenge wasn't how to code a templating engine, but which (what style of) tags to use in it. Thus, it was my realization to leave templating engines alone as PHP itself is a templating language of sort for HTML and thus the addition of a further layer on top of PHP would only hinder the speed of code as was proven by a simple code execution benchmark of Fari framework versus CodeIgniter. While the former displays majority of application's pages within 0.003 seconds, even the introductory page of the latter benchmarks at 0.0072 - 0.0079 seconds on the same setup[13].

8 Writing Applications in Fari MVC

Let's look at an exemplary use of Fari framework and how to write applications in it. As an example, let's say we want to use a **USER AUTHORIZATION** in our application, this should be a fairly common feature in a vast array of systems.

Once we have the framework in place (placed in a directory of our choosing), we need to decide which pages in our application will allow access to authorized users only. An example below shows that we will be using an *Admin_Controller* that contains an *index* action that we want to protect.

Thus we create our controller defining its functions as public if they are to be accessible from the Web:

```
class Admin_controller extends Fari_controller {
   public function index() {;} // secure admin index
}
```

A new way of authenticating users implemented in later versions of Kreus CMS makes use of preloading of functions as their parent controller is constructed:

```
// call authorization function if exists
if (is_callable(array($this, $authorize))) $this->$authorize();
```

Every developer can then define a function that is to be loaded when a new controller is constructed. In our case it is the following *Admin_Controller* function:

[12] The same method is employed by KOHANA FRAMEWORK; for more information see http://kohanaphp.com.

[13] http://kreus-cms.com eats up about 370 kb - 435 kb of system memory (on the server) under Fari MVC 0.7.6.

```
// by checking on construct from parent class we protect all
    functions (pages) in here
public function authorize_user() {
    if (users::check_session($_SESSION['user_id']) != true) {
        // change header (redirect) to login page if user_id not
             in session
        $url = 'Location: '.WWW_DIR.'/login/';
        header ($url);
        die();
    }
}
```

We set it and forget it, the authentication is then matter of the *Login_Controller* and related models.

9 Discussion

It is the sincere belief of mine that such a way of creating applications is great as it is harder to make mistakes.

Each group of pages we present to the user of our application needs a separate controller and each page we show needs a separate function within that controller . Whether a user is authenticated or not is done simply by writing a function that is called on controller construct.

After we make calls to models and process the result of these to the view registry, we 'close' each function by displaying it via view:

```
$this->registry->view->display('admin/admin_login');
```

If a part of the application is missing (such as is common during agile development), the user is either redirected to a default controller[14] in case a page is requested or if a model is missing[15] the developer is informed about that via a graphical output of the *Fari_Exception* class, you are almost glad you see an exception (Appendix F, Figure 33). There are more Questions & Answers pertaining to the framework direction in Appendix G.

The whole process can be captured as a mental model like such:

- group of pages (controller), authorize user (model) on construct
  - page (action)
    1. process data (model > view)
    2. view result (view > browser)

The clear advantage of the MVC pattern is its clear-cut approach to **separating each domain** of an application into a separate container [ZendCert06, p.162].

Hopefully, Fari framework will make a dent in the World and allow even beginner PHP programmers to develop applications in a more coherent way as is usual with languages like Ruby (Ruby on Rails) or Python (Django), it has helped the author of this project[16].

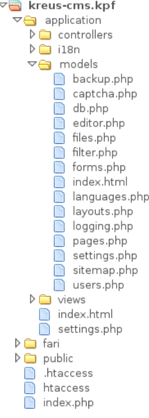
[14] Router is in charge of setting a default route or a 404.
[15] *Fari_Exception* class handles exceptions that come up during the initialization of the application.
[16] Example of which would be a TODO application PHPRODUCTIVITY created under Fari Framework in 30 minutes, acting in a same way as a Django application based on `http://net.tutsplus.com/tutorials/other/intro-to-django-building-a-to-do-list/`.

Part IV
CMS

10 CMS Architecture

Models used in Kreus CMS are shown in Figure 5.

Figure 5: Listing of models within Kreus CMS.



11 Database and Data Objects

It is in the nature of a content management system that a database is at its heart [PhpCms08, p.89].

11.1 The Problem

The vast majority of Web applications form some sort of a connection to a database. There is a wide variety of databases on the market and one often used with PHP development is the MySQL database[17]. However, as the databases used are wide and varied the framework should ideally support more than one type of a database.

The principle of **DECOUPLING** models and views also states that we should be building a model that can be **REUSED** and this model should be supported by an

[17] The M part of a LAMP stack.

Interface that can access more than one type of a database. So if an existing user wants to migrate from one CMS to another, they need to make sure the system will support their database.

Problems that need to be solved are:

- Handling of common queries, these queries to the database should be done efficiently and securely to ease development.
- Queries should be as platform independent as possible.
- Powerful data objects should be provided at a low cost [PhpCms08, p.89].

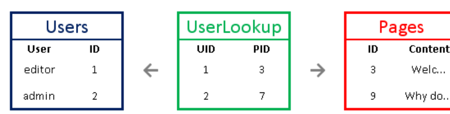
11.2 Framework Solution

11.2.1 Database Structure

The first problem we will need to consider when designing a database is how 'future proof' it is. When building a CMS we are storing pages in the database. These pages contain a unique identifier (*id*) and then a title, slug etc. Suppose we want to show, in further versions of the system, which articles were created by which user.

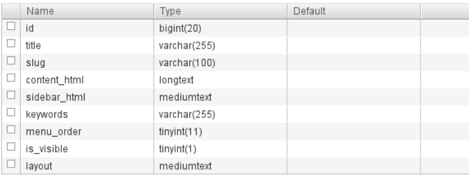
The solution is to use a **Relational Database** where a look up table serves as a connection between an identifier of one table and an identifier of a second table. In our example this would be a look up between user's *id* and page's *id*. This is what is called a **Many-to-Many Relationship** (Figure 6).

Figure 6: Many-to-many relationship in a relational database.



Currently however, the CMS does not need to store such information and thus a simple combination of three tables is implemented. If more fields need to be added to the *pages* table a new row is simply added to the table (Figure 7).

Figure 7: Database structure of the *pages* table.



Name	Type	Default
id	bigint(20)	
title	varchar(255)	
slug	varchar(100)	
content_html	longtext	
sidebar_html	mediumtext	
keywords	varchar(255)	
menu_order	tinyint(11)	
is_visible	tinyint(1)	
layout	mediumtext	

11.2.2 Handling Databases Easily with PDO

The PHP interface to databases provides for all the basic operations. But using the basic operations leads to a situation where similar code is written over and over again to handle common patterns of handling & processing. It is best to build these patterns into standard methods of the database handling class [PhpCms08, p.100].

To handle database code flexibly, we should decouple the application logic from the specifics of the database platform it uses [PhpOpp07, p.136].

PHP **PDO Extension** provides a single point of access for multiple databases. This loosens the coupling between the queries and the target database used. Its advantage over traditional methods[18] of creating database connections is that we don't have to modify our code if we change the database management system [Pdo07, p.7].

11.2.3 The Singleton Pattern

Well-designed systems generally pass object instances around via function calls. Each class retains its independence from the wider context, collaborating with other parts of the system via clear lines of communication. Sometimes, though, we find that this forces us to use some classes as conduits for objects that do not concern them, introducing dependencies in the name of good design [PhpOpp07, p.146].

An exemplary use is the database class that holds application-level information. We will use this object to use preferences such as DSN strings[19].

Our database class will need to be globally available (used by other models of the system) without the use of global variables that can be overwritten. When implementing this class we prevent it from being instantiated from outside of itself by defining its constructor as *private*.

```
private function __construct() { }
private final function __clone() { }
```

Defining *clone* function in this way enforces access solely via a *public* function [PhpCms08, p.159] that we will implement now.

```
1 private static $m_pInstance;
2 public static function connect($db_params) {
3    if (!self::$m_pInstance instanceof PDO) {
4       ... // try db connect
5    }
6    return self::$m_pInstance;
7 }
```

The *$m_pInstance* variable[20] stores the single instance of a database (accessible without needing an instantiation of the class) and is returned via a public interface using the *connect* function[21].

[18] PHP functions *mysql()* and *mysqli()*.
[19] Data Source Names hold table and user information about a database [PhpOpp07, p.146].
[20] The *$m_pInstance* property is private and static, so it cannot be accessed from outside the class.
[21] Because *connect* is public and static, it can be called via the class from anywhere in the code.

11.2.4 CRUD as a Database Abstraction Library

CRUD (or create-read-update-delete [Kilov09]) refers to the functions that should be implemented in a relational database application. Its implementation also works as a database abstraction library that makes it easier for application developers to implement SQL queries that tries to minimize errors.

The *db* class implements these (Appendix H.3, Figure 37) and as an example let us have a look at the *update* function within the model :

```
1 public static function update($table, $field, $value,
     $where_field, $where_value) {
2     // create SQL query
3     $sql = "UPDATE `$table` SET `$field`='{$value}' WHERE `
         $where_field` ='{$where_value}'";
4     // prepare statement
5     $statement = self::$m_pInstance->prepare($sql);
6     // execute statement
7     $statement->execute();
8 }
```

Consider thus the exemplary difference between the style we would write SQL queries normally and the use of the new CRUD interface:

```
SQL: "UPDATE pages SET published='1' WHERE title='products'"
db::update('pages', 'published', '1', 'title', 'products')
```

11.2.5 Discussion

It should be noted that a further extension of the CMS could be using **PAGE CACHING** where the pages to the site visitor are pre-prepared from when the content was converted to XHTML . This would mean further saving on CPU resources of the system as the pages would be served from a static copy rather than from a database. As for now, the current solution is fast enough.

12 Security

12.1 The Problem

Practically all software applications depend on some form of user input to create output. This is especially true for Web applications, where just about all output depends on what the user provides as input.

First and foremost, we must realize and accept that any user-supplied data is inherently unreliable and cannot be trusted [PhpSecu05, p.21].

One solution included in PHP is a 'MAGIC QUOTES' function but it misses its point when it creates problems [EssSecu06, p.90] especially for beginner PHP developers it is actually trying to help as we will later see.

Before the user can submit any data, we need to verify that the user is actually allowed to submit any content at all.

A special place needs to be left for file and folder handling where an incorrectly specified permissions can have dire consequences.

Thus we can cluster the security issues that need addressing into these categories:

- User access control.
- Input type checking.
- Character (quotes) escaping.
- File upload and folder handling.

12.2 Framework Solution

12.2.1 User Access Control

Many applications try to control who has access to what. Kreus CMS has an admin area for the content editor to use.

In order for the intended users to access the admin area, they need to be verified first. This is done via *users* class model .

The following check is thus made on admin area controller construct[22] to protect all of the implemented functions in one go:

```
(users::check_session($_SESSION['user_id']) != true)
```

The way Kreus CMS authorizes users is by checking whether *user_id* is set in the browser's *$_SESSION* [SecuPhp03, p.123]. *User_id* is stored in *$_SESSION* after a successful authorization from the *Login* page took place[23].

Login page containing a form requesting user input (*username* and a *password*) is also the first time we can expect a security flaw to come up as the data can be submitted from a custom form on the attacker's computer.

12.2.2 Form Token

One way of preventing submission of custom scripts is by using a **FORM TOKEN**.

Form token is generated and saved in the browser's *$_SESSION* each time login page is requested.

```
public static function create_token($token) {
   $_SESSION[$token] = md5(uniqid(rand(), true));
   return $_SESSION[$token];
}
```

This token is also saved in the view and submitted with the form as a hidden parameter.

```
<input type="hidden" name="form_token" value="<?php echo
    $form_token; ?>" style="display:none"/>
```

[22]As Router creates a new instance of a Controller, it checks for a defined authorization function.
[23]A process implemented later also makes sure that the *user_id* actually exists in the database.

Thus when authorizing the user we are also checking if these two tokens match.

```
if(isset($username, $password) && ($_SESSION['form_token'] ==
    $token))
```

12.2.3 Encryption of Data

Continuing from the example above if the user is successfully authenticated we save her *user_id*, as specified in the database model, into the *$_SESSION* with the added protection of encrypting this integer number by using **SHA-1** algorithm as a hash .

```
$_SESSION['user_id'] = sha1($user_id);
```

As passwords are stored in a database, they are also hashed using SHA-1 algorithm .

The starting point is that passwords should normally be handled using a one way hash function such as MD5 or SHA-1. Either is available as a function in PHP and more advanced variants collectively known as SHA-2 are likely to become available [PhpCms08, p.32].

Conceptually, the only workaround is for the hacker to generate a password that ultimately yields the same hash. With SHA-1 algorithm, this is extraordinarily difficult to do and requires an immense amount of time and resources to perform [PhpSecu05, p.142].

12.2.4 Type Checking

As has been stated at the beginning of the section, user input can not be trusted and thus any data sent from the user must be checked.

Kreus CMS uses a *filter* class model that handles all input checking and validation.

Let's take a look at an example of a page slug[24] converted from a page title that is submitted by the user and the relevant model function requested:

```
1  public static function to_slug($title) {
2     // convert to lowercase
3     $title = strtolower($title);
4     // replace diacritics
5     $title = str_replace(
6        array('e', 's', 'c', 'r', 'z', 'y', 'a', 'i'),
7        array('e', 's', 'c', 'r', 'z', 'y', 'a', 'i'), $title);
8     // strip the rest of diacritics & accents
9     $title = iconv('UTF-8', 'US-ASCII//TRANSLIT', $title);
10    // strip any non alphanumeric characters
11    $slug = preg_replace('/[^A-Z0-9 ,_\.-]/i', '', $title);
12    return $slug;
13 }
```

[24] Page slug serves as a name of the page requested by the browser.

1. We know that the page title has been already checked by the relevant model and thus we can only focus on the conversion to a slug format, that is, to lowercase.
2. As test users use Czech as their first language, the possibility of having diacritics in page titles has to be taken care of by stripping these characters off and only leaving the letters.
3. Then the next step is to verify if foreign characters were submitted. This can be accomplished using **REGULAR EXPRESSIONS**. Any character that is not alphanumeric, space or an underscore is left out.

Number checking can be done via a built-in function *ctype_digit* that returns true if a number is passed to it as a parameter.

```
public static function is_number($input) {
   return ctype_digit($input);
}
```

12.2.5 Character Escaping

On most Web hosts a *magic_quotes_gpc* option is enabled. This causes all *GET*, *POST*, and cookie variables sent to the application to be automatically filtered to "escape" special characters such as backslash '\' and the single and double quote characters. The intent of *magic_quotes_gpc* is to make strings safe for direct use in SQL queries to the database [PhpSecu05, p.182].

This option is intended to automatically protect code created in PHP and is targeted at beginner users. But even with this option on, single quotes are sometimes not escaped and a following error is then displayed:

```
SQLSTATE[HY093]: Invalid parameter number: no parameters were
    bound
```

Thus a solution is implemented via the *.htaccess* that turns magic quotes off system-wide. We can then focus on our own way of escaping input. Thus the *filter* class needs to be involved by escaping single quotes and other characters. The solution implemented in Kreus CMS is to change the ASCII character of a single quote to the relevant HTML character.

```
public static function escape($input) {
   // single quotes
   $input = preg_replace('/\x27/m', "&#39;", $input);
   // backslash
   $input = preg_replace('/\x5c/m', "&#92;", $input);
   $input = addslashes($input);
   return $input;
}
```

12.2.6 Securing Application Components

In order to make sure that only Fari framework can access the application code, all components of the system are secured by a variable check.

As we know, all access to the system is done via *index.php* file in the root directory and thus we can add a check if the framework is running or not to this file.

```
if (!defined('FARI')) define('FARI', true);
```

Thus the '*FARI*' variable is only defined when we access the system via *index.php*.

Each file, be it part of the framework or the application, then has a check implemented before any execution of the contained PHP code takes place.

```
<?php if (!defined('FARI')) {die(); } ?>
```

This, for example, also allows us to prohibit execution of Views without the appropriate Models . If '*FARI*' is not defined, the script dies in the browser.

Another way of stopping snoopy users in their tracks is to prohibit **FOLDER TRAVERSAL**. This can be done on an Apache Web server by including an *.htaccess* file, but as the file is hidden, it could potentially be left out by the application developer.

A simple solution, modeled after CodeIgniter, is to leave a blank *index.html* file in each directory. Thus this file is executed instead of the folder browse.

12.2.7 Securing File Access and Handling File Uploads

Some Web applications (sNews CMS comes to mind) ask of the users to change permissions on their uploads directory to 777. This causes even outside users to write to these files. An image file can then be changed to a PHP script and executed on the server.

A much more serious issue is posed by files created by PHP scripts. Since the Web server typically executes PHP applications, the Web server becomes the owner of all new files. This means that any other PHP script on the same Web server could potentially read and write those files. On shared hosting solutions, which are numerically the most common situations by far, any number of people can read and modify our data [PhpSecu05, p.136].

Thus Kreus CMS uses 0644 file permissions for the files, giving the user read and write permissions, and the rest only read permissions.

```
chmod($files_dir . $filename, 0644);
```

In addition, a folder allows to be executed by everybody so that we can traverse the folder and upload to it.

```
if (!mkdir($path.$dirname, 0755, true)) {
   return '<h3 class="no">'.$lang['Permission BAD'].'.</h3>';
}
```


12.2.8 Discussion

Cross-site scripting (XSS) is one of the most common vulnerabilities of web applications [PhpSecu05, p.53]. In such an attack, a hacker often stores JavaScript code in the application's database that is later executed when someone views it.

As an example, a successful attack[25] on Frog CMS can be seen in Appendix E, Figure 32. Kreus CMS doesn't directly work with the requested route but filters it through as it passes router. We then only request this array created by the router and can foil or at least minimize hacker's attempts. Kreus CMS *Login_Controller* has received a 'green light' from **Acunetix**[26] Web Vulnerability Scanner (Appendix E, Figure 31).

A few security **issues**, probably expected from an Open Source software are:

1. User credentials are sent in clear text.
2. Password input does not have auto complete switched off.

13 Content Backup

13.1 The Problem

One of the desirable functions in a CMS is creating and restoring backups of pages. The reason is two-fold.

One of the critical functions a CMS needs to have is **EXTENSION MANAGEMENT** as the system is useful when it can be easily extended [PhpCms08, p.11]. Using an MVC framework as a core of the CMS makes sure that the application itself is extensible. But the running system also consists of user created content which needs to be extensible or, more specifically, portable. The system needs a way of importing user created content from a 3rd party application and do the opposite transaction to a 3rd party application or for the same application on a different system, such as would be an upgrade to a different hosting solution where the system is based.

The second reason for implementing backups is a security one where content can be stored offline, on a different location from the Web host serving the application. Government and company regulations and policies regarding storage of data would fall into this category as well.

How do we make backups and more importantly, what format do we choose for content storage?

13.2 Framework Solution

13.2.1 XML

EXTENSIBLE MARKUP LANGUAGE (XML) has become the format of choice for communicating between disparate systems. XML is successfully used in many instances to store arbitrary data in a well-structured way [ZendCert06, p.171].

One of the useful aspects of PHP 5 is the way in which PHP handles XML data. The underlying code in the PHP engine was transformed to provide a seamless set of XML parsing tools that comply with World Wide Web Consortium (W3C) recommendations [ZendCert06, p.171].

[25] Has been reported on Frog CMS forum immediately.

[26] For more information see `http://www.acunetix.com/vulnerability-scanner`.

What we need is to create a structure where each page will contain elements like *id, title, keywords* etc (Appendix H.4, Figure 38).

```
1  <page>
2     <id>1</id>
3     <title>Home</title>
4     <slug>home</slug>
5     <content_html><h1>Kreus CMS</h1><p>Kreus CMS is a web
         application you can use to manage content of your site.
         It is primarily targeted at people who want to quickly
         update their small business websites and <i>first-time</i
         > CMS users.</p></content_html>
6     <sidebar_html><br></sidebar_html>
7     <keywords>kreus, cms, fari, framework, php</keywords>
8     <menu_order>1</menu_order>
9     <is_visible>1</is_visible>
10    <layout>page</layout>
11 </page>
```

PHP **DOM XML** functions follow a pattern. We create an object as either an element or a text node, add and set any attributes we want, and then append it to the tree in the spot it belongs to [PhpCook 02].

When creating a new XML file we first create a new DOM document with a root element '*pages*':

```
$dom = new DomDocument('1.0');
$pages = $dom->appendChild($dom->createElement('pages'));
```

A connection is then made to the database via *pages* model that retrieves the page listing to a variable; the passed parameter denotes that we want to get a listing of pages that are both visible on the site and also those that are just saved, still waiting to be published. This allows us to reuse the *get_list* function in different scenarios.

```
$list = pages::get_list("'1', '0'");
```

Forming XML is a matter of looping through the array in variable list and appending elements to the DOM document.

```
$page = $pages->appendChild($dom->createElement('page'));
$id = $page->appendChild($dom->createElement('id'));
$id->appendChild($dom->createTextNode($row['id']));
```

The resulting backup file is then saved to a directory of choice. This allows data to be **portable** on many systems.

When the user wants to restore her backup, we again create a DOM document, but this time get elements from the XML file to form a query that is then saved to the database.

```
$ids = $page->getElementsByTagName("id");
$id = $ids->item(0)->nodeValue;
```


13.2.2 Discussion

The interesting thing to note is that as data are not 'dumped' via an interface of a database administration tool (such as phpMyAdmin is for MySQL databases) the resulting file is easily readable by an end user and if the database structure were to change, only the parser in the *backup* model would need to be updated. The structure of the XML backup need not change.

As the CMS wants to be **extensible**, use of XML is appropriate as one of the major advantages of the DOM is that by following W3C's specification many languages implement DOM functions in a similar manner. Therefore, the work of translating logic and instructions from one application to another is considerably simplified [PhpCook 02, p.335].

14 Languages

14.1 The Problem

Information has to be stored in alternative versions for different languages, especially while computer translation remains a joke. So while some people may be able to do without it, many builders of a CMS will require language support [PhpCms08, p.201].

The issues that need to be considered are:

- Providing an interface to the end user in multiple language versions.
- Storing the language translations in an easy to edit solution.

14.2 Framework Solution

14.2.1 Handling Multiple Languages in Code

The information about a language is held in an instance of the *language* class combined with the *i18n* folder. This folder holds arrays of translations[27] in different languages.

```
return array(
   # login
   'Login' => 'Login',
   'Login Hover' => 'Login to the system',
```

The point here is to make the translations as flexible as possible to allow for concatenation of more words together. This allows the application developer to then form labels and messages to the end user in a flexible way such as:

```
return '<h3 class="yes">'.$lang['Directory'].$dirname.$lang['
    Created'].'.</h3>';
```

Languages class is yet another example of the **modularity** of the application where all messages to the user are handled by a model that loads files that can be easily updated by translators.

[27] English and Czech as of 15 Feb, 2009.

15 Search Engine Optimization

15.1 The Problem

In order to have a high probability of being accessed, a website should be in the TOP 30 references presented from a major search engine [Usability06, p.8].

Every website owner that wants to attract more visitors should not overlook the topic of **SEARCH ENGINE OPTIMIZATION** (SEO). Search engines use complex algorithms to rank pages in search results.

These algorithms are fed data from spiders that crawl the Internet in search of pages. These spiders, such as GOOGLEBOT then look for changes and updates that have happened on the site since their last visit. If, according to the algorithm, the site is updated more often, it has a bigger chance to appear higher (and earlier) in the search results [Seo05].

So, how do we make sure that the website managed with Kreus CMS is SEO optimized? After all, the CMS is targeted to small business owners and the goal should be the best site presentation.

15.2 Framework Solution

15.2.1 Optimization

Most typical Web users expect pages to download within 8 to 10 seconds at the max. The longer a person waits for a page to download, the more likely they are to have their stream of thought interrupted [Seo05, p.30].

- Kreus CMS achieves a split between the code and the design by having a separate directory for the views and a separate (public) directory for external content like Cascading Style Sheets (CSS) , JavaScript (JS) and images, thus these are not reloaded again while the visitor browses the site.
- The system does not use templating language and by doing so speeds up the load time of the site.
- The admin interface **CACHES** images that are to be loaded on subsequent pages and thus provides a more smooth interaction.

```
<div id="image-cache" style="display:none;">
   <img src="toolbar-icons.gif" />
</div>
```

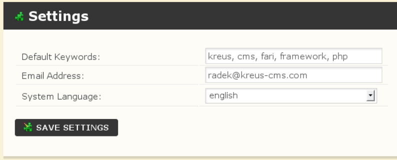
15.2.2 Keywords

Keywords are phrases the site owner wants to be found under. Keywords are what the prospective site visitors may type in a search box [Seo05, p.37].

Tags containing *meta-data* can influence search engines used for indexing of pages [PhpCms08, p.292].

- Kreus CMS encourages users to make use of keywords. The settings page contains an <*input*> field where keywords can be typed in (Figure 8). These will automatically appear whenever the user creates a new page; these can be further edited.

Figure 8: Site settings.



15.2.3 Sitemap

It is a good idea to have a sitemap [Usability06, p.62], linked to from the home page, which links to all major internal pages. The idea is to give search engine spiders another route through the site [Seo05, p.52].

- Fortunately, XML can be used in this case yet again. An official protocol for XML sitemaps exists at `http://www.sitemaps.org` with a detailed specification included. Thus DOM document is used again in the *Sitemap* class that generates a sitemap for the whole site. The protocol specification allows to specify the page's modification date and Kreus CMS utilizes this by always generating today's date as the *last-mod* parameter thus making the search engine spider think the site is **often updated** (which should be reflected in site rankings).
- When a new page is created in Kreus CMS, it can not-/appear in the main site menu. The *priority* tag in the XML document can be made use of by providing the (default) number/weight 0.8 for main pages (those that are in the menu) and giving a weight of 0.5 to the rest of the Web (those pages that are published yet not shown in the main menu).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
3    <url>
4       <loc>http://www.example.com/</loc>
5       <lastmod>2009-01-01</lastmod>
6       <priority>0.8</priority>
7    </url>
8 </urlset>
```

16 User Input

16.1 The Problem

We want the pages created via Kreus CMS to be **well-formed**. The site designer can take care of the layout but the page content is up to the content editor. As Kreus CMS targets non-tech-savvy people and first time CMS users how do we create well-formed XHTML documents without forcing the user to use XHTML tags when creating site content?

16.2 Framework Solution

16.2.1 Lightweight Markup Language

An initial solution (Appendix H.4, Figure 38) was to use a **LIGHTWEIGHT MARKUP LANGUAGE**.

A lightweight markup language is designed to be easy for humans to understand yet when parsed produces structurally valid XHTML.

TEXTILE[28] has been selected that takes plain text with *simple* markup and produces structurally valid XHTML code. It is used in Web applications, content management systems, blogging software and on-line forums.

As the first time user might not know the tags used in the target markup language a simple toolbar area has been implemented under each *<textarea>* element that handles user input. This toolbar contained a list of links that each had a JavaScript *onclick* action and a helpful tooltip [Ux07, p.498]. The *onclick* action was calling a function inside a pre-loaded *Textile.js* file. The parameter of this function was the formatting tag required, in this case *bold*. This script then took the position of the cursor in the *<textarea>* and added tags in front and behind it.

```
selection = a + article.value.substring(article.selectionStart,
     article.selectionEnd) + b;
```

A solution like this required a model implementation that parses markup language tags into XHTML . This was done via Textile class and as we would expect the class contained a bunch of regular expression search and replaces (reg-ex).

16.2.2 Rich Text Editor

A better solution that is easier to maintain and is more usable for the end user [PhpCms08, p.229] was to implement a **RICH TEXT EDITOR** (Appendix H.5, Figure 39).

These editors, most commonly built using JavaScript, need to provide content to and from users at appropriate times [PhpCms08, p.229].

MOZILLA FOUNDATION[29] page contains specifications on the implementation of the *designMode* in modern web browsers so that we can turn an *<iframe>* element[30] into an editable area.

An implementation of a lightweight RTE (Figure 9) works in these steps:

1. Content that needs to be pre-loaded into the editor is output into a JavaScript variable contained in the *<head>* section of the Admin template.

2. This variable is then written into a target *<iframe>* as per Mozilla specification.

3. The *<iframe>* is set as editable via a *designMode* so that we can execute commands on it.

4. Links in the editor toolbar call an *onclick()* function that executes a function in an external JavaScript file setting a text in the editor as bold etc.

[28] For more information visit `http://textile.thresholdstate.com/`.

[29] For the specification document visit `http://www.mozilla.org/editor/midas-spec.html`.

[30] *<iframe>* element represents a separate HTML page with its own head and body.

Figure 9: Editing a page content in Kreus CMS.

16.2.3 Discussion

An interesting solution has been applied to the Highlight button in the RTE.

The MOZILLA specification can apply a forefront color to a selected text that could serve as a form of highlighting but this forces the user to always use the same color of highlighting with Kreus CMS.

To allow for flexible interpretation of the "Highlight" command *editor* model class is called when a page is to be displayed to the site visitor. A *<font color>* tag as produced by the editor is then replaced by a *<span>* tag instead and thus the design decision is again left to the site owner.

Another issue however arises. As our RTE works with the same HTML code that is to be displayed to the site visitor and site editor, how would the RTE interpret the *<span>* tag? This could be fixed in two ways:

1. Convert the *<span>* tag back to its original *<font color="#996633">* form.

2. Hardcode a style information for the *<span>* tag in the RTE[31].

Neither of these are very flexible and increase on workload and clutter. Thus, the "cleanup" class is only called when a page is requested by the site visitor and the result is only viewed, not saved. This allows for further replacements, like changing a textual (TM) to a trademark tag in HTML etc[32].

Enforcing content editor (the person) to only use buttons that create prepared tags is also beneficial from a security perspective as we often know what input to expect[33] and there is no need to involve a 3rd party library that would have to cleanup our code [PhpCms08, p.230].

17 Site Content

17.1 The Problem

Ultimately, CMS is used to manage content of a site. Thus a special chapter needs to be left reflecting on the content delivery used.

Each CMS has its own way of separating content. Blogs usually consists of articles and relating comments, static pages are split into different areas and categories

[31] Write a style information in the *<head>* of the *<iframe>*.

[32] The editor leaves a *
* tag behind even if no text is entered into the *<iframe>*.

[33] TinyMCE, a widely used RTE, allows for a direct use of HTML tags.

based on the template used and let's not forget extensions to the basic CMS model that enhance the interactivity like RSS feeds, news section or a contact form.

It is also pretty much essential to have an ability to create items of this kind in an unpublished state so that they can be revised until ready for use [PhpCms08, p.282].

As this is the 'bread and butter' of the CMS, these are the issues needing consideration:

- How do we form a **structure** of the site managed that is relevant to the real world application while being easy to understand by the target audience?
- How do we manage the state of content that can be **published**/unpublished?
- How to structure extensions to enhance the interactivity of the site thus making the site more **usable**?
- As content is submitted via **FORMS** , how do we make sure that the user does not have to retype all of the fields in a form if the form is not successfully submitted?

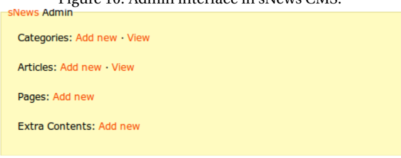
The final point we set to ourselves was the implementation of a contact form (Appendix H.4, Figure 38) which, is assumed, would want to be used by small businesses very often.

A multitude of control-laden dialog boxes does not make a good user interface [Ux07, p.440]. We need to decide the little details so that the end users don't have to [37sig06, ch.6].

17.2 Framework Solution

17.2.1 Content Structure & Editing

Figure 10: Admin interface in sNews CMS.



[Ux07, p.25]: developers and marketers often use the language of features and functions to discuss products. This is only natural. Developers build software function by function, and a list of features is certainly one way to express a product's value to potential customers (though this is clearly limiting, as well). The problem is that these are abstract concepts that only provide limited insight into how human beings can be effective and happy while using technology.

Do beginner users think in terms of categories, articles, pages and extra contents as implemented by sNEWS CMS (Figure 10)?

Instead of relying on any assumptions about users a **LOW LEVEL PROTOTYPE** of the system was made and tested for interactivity and ease of use by target users. The result of this study was that users felt 'at home' with a simple page item that is split

into 'main content' and an 'extra content'. As the extra content is often used as a sidebar, users felt (Appendix H.2, Figure 36) it should be called as such (Figure 11).

The other issue that needed to be considered is how to implement a main/substructure of the content?

Often CMS uses a categories table storing a list of different categories with different depth in the page structure. Fortunately we can get away with a category table by simply stating whether a page should be visible in the menu or not, the same way we can set a page as published/unpublished. As the intended audience are small business users and especially first time CMS users, any semantic split of the content could be detrimental. This is interesting as implementing category should be simple via a relation database, it is not that we can't do it, it's that we don't want to.

Figure 11: Editing a page in Kreus CMS.

17.2.2 Serving Pages

Pages model works in conjunction with pages controller much like other components of the system work. We make use of the *route* array as stored in all controllers.

```
1 // page name provided
2 if (!empty($this->route[1])) {
3    // route to the sitemap thru here
4    if ($this->route[1] == 'sitemap.xml') { self::sitemap(); }
5    else {
6       // get the page requested
7       $page = pages::get_page($this->route[1], "'1'");
8       // if no page returned (page does not exist) get homepage
```

```
9        if (empty($page)) { $page = pages::get_home(); }
10    }
11 // request page that is first in the menu (homepage)
12 } else { $page = pages::get_home(); }
```

17.2.3 Discussion

An interesting challenge is how to determine which page to load if the *index* is requested. As we have explored previously the MVC framework uses a router which also splits the incoming route and loads appropriate controller and action but this was the case when we manually define which controller/action combination will work as a default one, system wide.

To provide for a **flexibility** in having different pages serving as a homepage[34] while again not confusing the user with another option that serves as a 'homepage selector', the pages class determines which page should serve as a homepage by itself.

The solution becomes obvious when we realize that a homepage is a page that is the first most in the menu while having a status of being published at the same time.

```
1 public static function get_home() {
2    try {
3       // visible and in menu; sorted by lowest menu order
4       $result = db::raw_select("SELECT title, content_html,
          sidebar_html, keywords FROM pages WHERE is_visible = 1
           AND       menu_order != 0 ORDER BY menu_order ASC");
5       ...
6    } catch(Exception $e) { echo $e->getMessage(); }
7 }
```

17.2.4 Generating a Secure Contact Form

Implementation of a contact form is interesting from the front end view as instead of cluttering the interface with control structures, a contact form is generated as a last item in the site menu if the admin/content editor saves her email address on the settings page.

A content form is generated using two models : *forms* class and a *captcha* class.

A **CAPTCHA** is a program that generates and grades tests that are human solvable, but intends to be beyond the capabilities of current computer programs. This technology is now almost a standard security mechanism for defending against undesirable or malicious Internet bot programs, such as those spreading junk emails and those grabbing thousands of free email accounts instantly [Yan08].

As numerous mechanisms have been implemented that break the most widely used captchas, it has been decided that a new captcha generator needs to be implemented, one that is novel enough (for now at least).

[34] Frog CMS, for example, has an undeletable pre-set homepage.

Image Captcha

The interesting part is how PHP generates images. The appropriate view (contact page) template contains a link to an image that acts as yet another page. Thus the *Pages_Controller* handles its request.

```
<img src="<?php echo WWW_DIR; ?>/pages/captcha_image/" />
```

```
public function captcha_image() { captcha::create(); }
```

The relevant function in the *captcha* model then sets the header of the 'page' to PNG image and adds the characters we want the visitor to check for.

```
header('Content-type: image/png');
```

However, there are drawbacks to using captchas, as they are an irritating overhead for legitimate site visitors, and can be difficult or even impossible to read, which goes against principles of accessibility [PhpCms08, p.36]. What the previous quote does not mention is the fact that as captchas consist of text on an image, they won't be displayed on a browser that has loading of images disabled and won't be understood by users that have no other means of controlling their computers other than by voice.

17.2.5 Sending Email

As our contact form needs a way of sending email messages to the site owner a decision needs to be made whether to send messages via PHP's *mail()* function.

Its advantage quickly vanishes when our Web host informs us that this function is disabled[35]. The solution, then, is to use **PEAR'S MAIL** class [PhpCook 02, p.448] that sends messages via SMTP protocol. *PEAR::Mail* also checks whether email addresses it is supposed to use exist, this adds an extra protection against SPAM bots that submit non-existent email addresses in the 'From' field.

17.2.6 Usable Forms via Ajax

When forms are submitted (sent for processing) in XHTML an *event* fires on a DOM element and this takes us to another page where the form is processed. If we were to return to the form back, the content of it would be gone. Let's imagine that a content editor tries to submit his or her work and specifies a title that contains special characters, as this means the page isn't saved, she or he would loose all of his or her work.

Using **MOOTOOLS** library and its functions we can stop the default behavior [MooTools08, p.74] that happens when the form is submitted and process the content of it without needing a page reload.

[35] As PHP is executed in a shared hosting environment under user *nobody*, *mail()* function is often misused for sending SPAM. It is also used by Nigerian royalty.

```
$('newPageForm').addEvent('submit', function(e) {
   // prevent the submit event
   new Event(e).stop();
   ...
```

As the user might not know whether the page is being saved or not we need to provide him or her with a feedback [Usability06, p.18]; we attach a class to a defined *<div>* element that contains spinning GIF image to show that processing occurs.

```
var log = $('form_result').empty().addClass('ajax-loading');
```

We than remove the spinner after we have a result of the processing, a message telling the user what is the result of her or his action. As redrawing and re-rendering of information should be minimized [Ux07, p.179], using asynchronous JavaScript makes a Web application more usable making users feel that they are in an environment, not that they are navigating from page to page or place to place.

Part V
User Evaluation

18 Introduction

How do we know if we were successful in creating a simple CMS for beginner users? We ask them, after letting them (5 people) use the system for a month with live support.

Following is a measurement of quality of software in terms of a collection of attributes that have stood the test of time [FactsEng03, ch.3].

18.1 Evaluation of Software Quality

The ISO 9126 standard prescribes characteristics that describe software quality [WebEng05, p.113]. Its variation **ISO 9126-2:2003** specifically explains how to apply international software-quality metrics. Following are six requirements for evaluating software products [CISA08, p.258]:

1. **Functionality** of the software processes that satisfy implied needs.
2. Ease of use (**usability**) meaning the effort needed for use by a target set of users.
3. **Reliability** with consistent performance for a stated period of time.
4. **Efficiency** of resources or performance relating to resources used.
5. **Portability** between environments, transfer from one system to another.
6. **Maintainability** with regard to making modifications or effort needed to make a modification.

The standard specifies four rating levels for each of these requirements: *excellent, good, fair* and *poor*. The goal of the evaluation is to rate user experience with the system based on these requirements and levels. Questions asked are kept short and addressing only one area at a time [Users04, p.269].

19 Results

Figure 12: Kreus CMS software quality assessment results.

requirement	rating
Functionality	excellent
Usability	good
Reliability	good
Efficiency	excellent
Portability	good
Maintainability	excellent

The largest number of suggestions for changes to a Web application are design related [SoftEng06, p.146].

Questions given to end users and consequent graphs charting user's level of satisfaction are shown in Appendix A. Based on these findings (Appendix A, Figure 14), we can conclude that the software fares well, Open Source perhaps being the saving grace.

Furthermore, we can assess two requirements, portability and maintainability based on the development process and target platform used.

The assessment results are shown in Figure 12 and detailed further below.

Functionality *excellent*

Functionality answers that Kreus CMS 'does the job'. This is perhaps not surprising as the requirements set out before writing this software (Subsection 3.2) were lean and the focus was on 'doing the job' rather than on providing loads of functions that will only be used little.

The system simplifies managing of content for small business websites.

Usability *good*

Appropriateness rates the lowest in the usability category especially with a bit more advanced users perhaps hinting at the fact that some functions and screens are very simplified and don't allow for customization. The system was developed so as to fine tune these as the system goes through the testing and users reflected on that.

It is a relief that users understand the system.

Reliability *good*

Again, the system has been released early and a lot of errors were fixed during the nearly two-month testing period. Users were instructed to also take into account the errors they saw throughout the testing.

A good suggestion might be to write an exception model working only in the Kreus CMS domain to provide a consistent way of displaying error messages, the same way Fari MVC reports if something goes wrong (Appendix F).

Efficiency *excellent*

Efficiency scores highly as the result of Kreus CMS running in a Web browser. Also, no gigantic content is served as could happen with news agencies etc.

Kreus CMS is effective thanks to its intended use and being executed on the Web server.

Portability *good*

Web applications have the advantage of running in a Web browser thus we can do without a specific operating system. Kreus CMS has been tested in current major browsers (Appendix D[36]) and accomplishes this task. However there are some limitations as to which Web hosting solutions can be used. The advantage of using PDO as a database handler makes the system's database very portable, yet if PDO is not

[36]Browser testing accomplished under `http://xenocode.com` running under Windows Server 2008 running in Sun VirtualBox under Linux Mint.

provided by the Web host, no database connection can be made[37]. The same can be told of *PEAR::Mail* email handling function that has not been specifically written for Kreus CMS and thus we rely on external 3rd party libraries[38]. Finally, the RTE needs to have JavaScript on. We could create a fallback where the content would be output to a *<textarea>* with HTML tags throughout, but keeping in mind the target audience, this could do more harm than good.

Maintainability *excellent*

Using a lean MVC framework means an easy addition and updating of modules. The system has been documented from the very beginning and thus its parts should be easy to understand. We extend on this fact in the last chapter.

19.1 Discussion

An interesting dilemma arose during the evaluation of the system by users a dilemma concerning the **separation of concerns**.

We, as Computing people understand the term and try more and more to achieve the separation of style and content, be it by using HTML and CSS, XML or even by employing frameworks such as Fari MVC. To the average person, however, this theme is often quite alien. Some people (one person during the evaluation in particular) feel that it is a disadvantage of the CMS when they can't "change color of the heading, of the text, more highlighting styles, border around the text, background image...". This theme often makes its way into geeky comics but when you are a person managing a CMS for a customer, it is far from amusing.

How should one act then? Should we give in and let the users do whatever they want with the site (so that it looks like an average MySpace profile) and get bonus points for having 'happy' users?

> A mental note for the future would be to always educate users as to why some things are discouraged and how a modern site looks like. In the end, users should know that CMS stands for Content management and not site management.

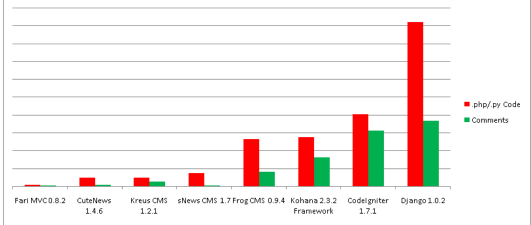
[37]A custom PDO class would have to be written.
[38]Kreus CMS tries to send an email via *mail()* if *PEAR::Mail* is not provided.

Part VI
Conclusions

Previous chapters were written so as to summarize the problems and solutions encountered and implemented during the building of a CMS, describing its stages as building the user interaction, MVC framework and the application itself. This part will focus on thoughts relating to future work.

20 Future of Kreus CMS

Figure 13: Comparison of code and comments size for Frameworks and CMS.



Having an extensive enterprise level Web application is one thing but understanding it is second. Whether it is a small business owner trying to use a CMS for the first time or a PHP developer trying to create an application on a new framework, how can we expect them to succeed in their efforts if a clear to understand documentation is not provided?

A standard in the IT industry used to be writing a DOC STRING explaining which functions in an application do what, and how variables are passed around.

However, these are useful to a seasoned developer yet not to a programmer new to the system. Certainly thousands of lines of doc strings aren't going to convince any programmers to **adopt** a piece of software. It is much more important to present clear English prose[39] that demonstrates the quality of someones thinking and design work in attacking a real problem [SoftEng06, p.145]. Thus when talking about a future of any system we need to make sure that the application has programmers who are willing to adopt it. Personally, my aim was to bridge the gap for not only beginner CMS users, but for semi-advanced PHP developers also. We can, at least, look at the result of this aim by overviewing the size of code[40] and a relation between the number of lines of comments and lines of codes for frameworks and CMS (Figure 13).

[39] The coding and documentation style comes closest to CodeIgniter, for more information see `http://codeigniter.com/user_guide/general/styleguide.html`.
[40] As calculated by CLOC-1.07, for more information see `http://cloc.sourceforge.net/`.

Please note that this is in violation of [AgileSw08, p.54] arguing that a lot of comments represent a failure of the programmer as these can not be maintained. So far they are.

User Requirements

We also can not forget the fact that system requirements for Kreus CMS revolve around assumptions about users' needs and wants. An application developed for a 3rd party client would probably have to follow a more rigid set of user requirements methods, tools and techniques.

Making Features Work Hard to be Implemented

Borrowing from 37signals, if every feature that anyone requests was implemented, it would make an unusable product [37sig06, ch.5].

Page Caching

This is not to say that extensions are not possible, one example might be, as discussed in the previous chapters, **page caching** where a page for the site visitor is served from a static copy. An assumption can be made that an end user of Kreus CMS won't be creating new content faster than a news agency or a major blog, thus it is safe to assume that such a feature would be welcomed. It does not affect end users in a negative way and only increases speed of site execution.

Session Injection

Another feature that would fall into the 'back end' category is the protection of session. Currently, the *user_id* is stored in the user's session, encrypted, but that is not enough [EssSecu06, p.82]. Storing sensitive data in this way leaves the system open to **session injection**. The best solution would be to store this data in the database.

Global Input Escaping

Frameworks such as CodeIgniter or Symfony escape all *$_POST* and *$_GET* input and store it in a model for use by other models. Implementing a similar method (such as already is done by cleaning input route and storing it in all controllers) would count to another one of these 'set it & forget it' functions that help make the system more secure.

Mental Model

The purpose of using Ajax on form submit has been explored, it is to prevent reloading of pages. However, we might actually want to redirect the user after a form submit has been successful. An example of which would be saving a page into database after its content has been validated[41,42]. This would mean integrating JavaScript

[41] Interestingly, no one has requested this feature yet.
[42] This could be implemented by checking the response of a query and then redirecting via *window.location*.

form submission file into PHP[43] as we need to dynamically set the target page after the redirect and not hard code it.

Target Audience

In the end, Kreus CMS needs to stay true to its **target audience** rather than trying to be all things to all customers[ExtremeProg03, p.4]. An example of a CMS that broke this rule is sNews CMS that, with the addition of a new version, has reworked and increased the number of types of content that can be published and made the interface less usable with the addition of options and settings everywhere.

But as it is understood that different architects have different (and opposing views) on how things should be done[FactsEng03, ch.2], Kreus CMS is **open source** and allows for branches to be created. This should, hopefully, create a bigger market of CMS targeted to a variety of uses.

Fari Framework

We can not forget on the spinoff project created as a tool on which Kreus CMS runs, that is Fari MVC Framework. An aim, perhaps as an MSc. project, is to make use of **MODEL DRIVEN ARCHITECTURE** and code generation and create a platform that generates a Web application from its model. An example could be a BOOSTER model [Mda05] developed by University of Oxford that has been used in many real life scenarios. The extension would not only create a code in a specified language but would target it for a specific framework such as CodeIgniter or Fari. Such a venture, when properly executed[44], would be an exciting addition to the agile world of Web development.

Closing Remark

Building an MVC Framework and a CMS proved to be a task that can be continuously mined for new ideas and improvements. It is not a project that you build and are dealt with. After you do one part you realize you could have done better so you fine tune some more and your level of expertise rises which leads to further ideas of what could be done, which is great.

[43]This is done by setting *header('Content-type: application/javascript');* in the PHP file.
[44]The generated code needs to be updatable when the model changes.

Appendixes

Credits

Giving credit where credit is due, work of these authors is included with the final distribution of Kreus CMS.

- YUSUKE KAMIYAMANE, *Fugue Icons* used in the Admin interface, released under Creative Commons Attribution 3.0 License[45].
- MARK JAMES, *Silk Icons* used in the Admin interface, released under Creative Commons Attribution 2.5 License[46].
- VALERIO PROIETTI, *MooTools* compact JavaScript framework, released under Open Source Initiative MIT License[47].

[45]For more information visit http://www.pinvoke.com/ (retrieved 16 Feb, 2009).
[46]For more information visit http://www.famfamfam.com/lab/icons/silk/ (retrieved 16 Feb, 2009).
[47]For more information visit http://mootools.net/download (retrieved 16 Feb, 2009).

Glossary[48]

A

Ajax (**A**synchronous **Ja**vaScript and **X**ML) is a group of interrelated Web development techniques used for creating interactive Web applications or rich Internet applications.

Apache A very popular Open Source, Unix-based Web server.

C

CAPTCHA (**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part) A category of technologies used to ensure that a human is making an online transaction rather than a computer.

Chmod The chmod command (abbreviated from **ch**ange **mod**e) is a shell command in Unix and Unix-like environments. When executed, the command can change file system modes of files and directories.

CMS (**C**ontent **M**anagement **S**ystem) Software that is used to create and manage the content for a Web site. It provides for the storage, maintenance and retrieval of HTML and XML documents and all related image, audio and video files.

CodeIgniter is an open source Web application framework for use in building dynamic Web sites with PHP.

CRUD (**C**reate, **R**ead, **U**pdate and **D**elete) are the four basic functions of persistent storage, a major part of nearly all computer software.

CSS (**C**ascading **S**tyle **S**heets) is a stylesheet language used to describe the presentation of a document written in a markup language.

D

Django is an open source Web application framework, written in Python, which loosely follows the model-view-controller design pattern.

DOM (**D**ocument **O**bject **model**) is a platform- and language-independent standard object model for representing HTML or XML and related formats.

DSN (**D**ata **S**ource **N**ame) is a name of a data structure that contains the information about a specific database that an Open Database Connectivity (ODBC) driver needs in order to connect to it.

F

Frog CMS is an open source content management system originally developed by Philippe Archambault.

[48] Provided mostly by `http://www.answers.com`.

G

GET is an HTTP command used to request a file from a Web server.

J

JavaScript is a popular scripting language that is widely supported in Web browsers and other Web tools.

M

MD5 (**M**essage **D**igest **5**) A popular one-way hash function developed by Ronald Rivest.

MDA (**M**odel-**D**riven **A**rchitecture) is a software design approach for the development of software systems. It provides a set of guidelines for the structuring of specifications, which are expressed as models.

MVC (**M**odel **V**iew **C**ontroller) An architecture for building applications that separate the data (model) from the user interface (view) and the processing (controller).

MySQL is a very popular open source, relational DBMS for both Web and embedded applications.

O

Open Source Refers to software that is distributed with its source code so that end user organizations and vendors can modify it for their own purposes.

P

PDO (**P**HP **D**ata **O**bject) is an extension that defines a lightweight, consistent interface for accessing databases in PHP.

PEAR (**P**HP **E**xtension and **A**pplication **R**epository) is a repository of PHP software code.

PHP (**P**HP **H**ypertext **P**reprocessor) A scripting language used to create dynamic Web pages.

PNG (**P**ortable **N**etwork **G**raphics) A bitmapped graphics file format endorsed by the World Wide Web Consortium.

POST is an HTTP command used to send text to a Web server for processing.

Python is a general-purpose, high-level programming language. Its design philosophy emphasizes programmer productivity and code readability.

R

Ruby is an interpreted, object-oriented programming language that is somewhat similar to Perl in syntax.

RoR (**R**uby **o**n **R**ails) is an open source Web application framework for the Ruby programming language.

RTE **R**ich **T**ext **E**ditor presents a "what-you-see-is-what-you-get" interface for editing rich text within web browsers. The aim is to reduce the learning curve associated with users trying to express their formatting directly as valid HTML Markup.

S

SEO (**S**earch **E**ngine **O**ptimization) Designing a Web site so that search engines easily find the pages and index them.

SESSION is a semi-permanent interactive information exchange between a computer and a user.

SHA-1 (**S**ecure **H**ash **A**lgorithm-**1**) A popular one-way hash algorithm used to create digital signatures.

SMTP (**S**imple **M**ail **T**ransfer **P**rotocol) is a standard e-mail protocol on the Internet and part of the TCP/IP protocol suite, as defined by IETF RFC 2821.

sNews is a single file Content management system original developed by Luka Cvrk.

T

Textarea is an on-screen rectangular frame into which you type text.

Textile is a lightweight markup language originally developed by Dean Allen and billed as a "humane Web text generator".

U

URL (**U**nified **R**esource **L**ocator) An Internet address.

W

W3C (**W**orld **W**ide **W**eb **C**onsortium) An international industry consortium founded in 1994 by Tim Berners-Lee to develop standards for the Web.

WYSIWYG (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et) It refers to displaying text and graphics on screen the same as they will print on paper or display on a Web page.

X

XHTML (e**X**tensible **HTML**) A markup language for Web pages from the W3C.

XML (e**X**tensible **M**arkup **L**anguage) An open standard for describing data from the W3C.

XSS (**CROSS**-**S**ite **S**cripting) Causing a user's Web browser to execute a malicious script.

References

[SoftEng06] Andersson, E., Greenspun, P., & Grumet, A., (2006). *Software Engineering for Internet Applications*. Cambridge: MIT Press.

[SoftArch09] Monson-Haefel, R., (2009). *97 Things Every Software Architect Should Know: Collective Wisdom from the Experts*. Sebastopol: O'Reilly.

[FactsEng03] Glass, R., (2003). *Facts and Fallacies of Software Engineering*. Boston: Addison-Wesley.

[RapidDev96] McConnell, S., (1996). *Rapid Development*. Redmond: Microsoft Press.

[ExtremeProg03] Wallace, D., Raggett, I., & Aufgang, J. (2003). *Extreme Programming for Web Projects*. Boston: Addison-Wesley.

[AgilePract08] Man Lui, K., & Chan, C.C.K. (2008). *Software Development Rhythms: Harmonizing Agile Practices for Synergy*. New York: Wiley-Interscience.

[AgileSw08] Martin, R., (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River: Prentice Hall PTR.

[ProPhp08] McArthur, K., (2008). *Pro PHP: Patterns, Frameworks, Testing and More*. Berkeley: APress.

[OoPhp06] Lavin, P., (2006). *Object-Oriented PHP: Concepts, Techniques and Code*. San Francisco: No Starch Press.

[PhpOpp07] Zandstra, M., (2007). *PHP Objects, Patterns, and Practice, Second Edition*. Berkeley: APress.

[PhpCms08] Brampton, M., (2008). *PHP 5 CMS Framework Development*. Birmingham: Packt Publishing.

[CodeIgnit07] Upton, D., (2007). *CodeIgniter for Rapid PHP Application Development*. Birmingham: Packt Publishing.

[PhpCook 02] Sklar, D., & Trachtenberg, A., (2002). *PHP Cookbook*. Sebastopol: O'Reilly.

[ZendCert06] Shafik, D., & Ramsey, B., (2006). *php|architect's Zend PHP 5 Certification Guide*. Toronto: Marco Tabini & Associates, Inc.

[Pdo07] Poppel, D., (2007). *Learning PHP Data Objects*. Birmingham: Packt Publishing.

[PhpSecu05] Alshanetsky, I., & Lerdorf, R., (2005). *php|architect's Guide to PHP Security*. Toronto: Marco Tabini & Associates, Inc.

[EssSecu06] Shiflett, C., (2006). *Essential PHP Security*. Sebastopol: O'Reilly.

[SecuPhp03] Kabir, M., (2003). *Secure PHP Development*. Indianapolis: Wiley Pub.

[RegexPkt07] Stubblebine, T., (2007). *Regular Expression Pocket Reference, Second Edition*. Sebastopol: O'Reilly.

[MooTools08] Newton, A., (2008). *MooTools Essentials: The Official MooTools Reference for JavaScript™ and Ajax Development*. Berkeley: APress.

[Usability06] Koyani, S. J., Bailey, R. W., & Nall, J. R., (2006). *Research-Based Web Design & Usability Guidelines*. Washington: U.S. Department of Health & Human Services.

[Ux07] Cooper, A., Cronin, D., Reimann, R., & Cronin, D. (2007). *About Face 3: The Essentials of Interaction Design*. Indianapolis: Wiley Pub.

[WebEng05] Mendes, E., & Mosley, N. (2005). *Web Engineering: Theory and Practice of Metrics and Measurement for Web Development*. Berlin: Springer.

[CISA08] Cannon, D., (2008). *CISA: Certified Information Systems Auditor Study Guide*. San Francisco: Sybex.

[Users04] Courage, C., & Baxter, K. (2004). *Understanding Your Users: A Practical Guide to User Requirements*. San Diego: Morgan Kaufmann.

[Seo05] Wall, A. M., (2005). *Search Engine Optimization Book*. State College: Aaron Matthew Wall.

[Mda05] Davies, J., Crichton, C., Crichton, E., Neilson, D., & Sørensen, I. H., (2005). *Formality, Evolution, and Model-driven Software Engineering*. Oxford: Oxford University Computing Laboratory.

[Yan08] Yan, J., (2008). *A Low-cost Attack on a Microsoft CAPTCHA*. Newcastle: Newcastle University.

[Kilov09] Kilov, H., (1990). *From Semantic to Object-oriented Data Modeling*. New Jersey: Bell Communications Research.

[Nemetral08] Nemetral, (2008). *A Gentle Introduction to MVC*. Retrieved Feb 15, 2009, from: `http://nemetral.net/2008/07/31/a-gentle-introduction-to-mvc-part-1/`.

[Elliot07] Elliot, E., (2007). *PHP Tip: Extract, Variable Variables and Templating*. Retrieved Feb 15, 2009 from: `http://www.ejeliot.com/blog/101`.

[37sig06] 37signals, (2006). *Getting Real: The Smarter, Faster, Easier Way to Build a Successful Web Application*. Retrieved Feb 18, 2009 from `http://gettingreal.37signals.com/toc.php`.

Index

.htaccess, 21, 23, 35

404, 24

Abstract Function, 24
Agile Development, 14, 15, 27
Ajax, 16, 46
Apache Web Server, 21, 35
Authorization, 26, 32
Autoloading, 22

Backup, 36

Caching, 31
CAPTCHA, 45
Character Escaping, 34
Clone, 30
CMS, 28
CodeIgniter, 21, 26, 35
Contact Form, 16, 43, 45
Controller, 21, 24–27, 32, 44–46
CRUD, 31
CSS, 39, 42

Database, 28–30
Database Abstraction Library, 31
Django, 27
DOM, 37, 38, 40, 46

Encryption, 33
Exceptions, 22

File Access, 35
File Uploads, 35
Forms, 16, 43, 45, 46

Hash Function, 33

JavaScript, 16, 39, 41

Keywords, 39

Languages, 38
Login, 32
Low Level Prototyping, 43

Magic Methods, 22, 23, 25
Magic Quotes, 31, 34
Many-to-Many Relationship, 29
Markup Language, 41
MD5, 33
Model, 17, 20–22, 25, 27, 28, 30–33, 35, 37, 38, 41, 42, 44, 45
Model-View-Controller, 14, 16, 20, 22, 36, 45
MySQL, 28, 38

PDO, 30
phpMyAdmin, 38
Python, 27

Quotes Escaping, 34

Registry, 22, 23, 25
Regular Expressions, 34, 41
Relational Database, 29, 31, 44
Requirements, 15
Rich Text Editor, 41
Router, 23, 24, 27, 36, 45
Ruby, 27
Ruby on Rails, 27

Security, 31, 42
SEO, 16, 17, 39
SESSION, 32
SHA-1, 33
SHA-2, 33
Sharing Hosting, 35
Singleton Pattern, 30
Site Content, 42
Sitemap, 16, 17, 40
sNews CMS, 35, 43
SQL, 31, 34

Templating, 21, 25, 26, 39
Textile, 16, 41
Texy, 16
Three State Solution, 19
Token, 32
Type Checking, 33

URL Rewriting, 21
User Access Control, 32
User Evaluation, 48
User Input, 40
User Interface, 19

View, 15, 20, 21, 25, 27, 28, 32, 35, 39, 46

W3C, 36, 38

XHTML, 31, 40, 41
XML, 36–38, 40
XSS, 36

Zend Framework, 22

A System Evaluation

A.1 Questionnaire

About End Users

1. What is the purpose of your site.
2. How often do you edit content of your site: daily, weekly, monthly, yearly.
3. What is your main method of managing content: CMS / Web designer / WYSIWYG editor.
   (a) If CMS: Which?
4. Have you heard about CMS already? yes / no
   (a) If yes: What do you think a CMS should do for you?
   (b) If no: What do you think a Web application that lets you create and edit site content should have?

About CMS

Mark satisfaction of following items, as they relate to Kreus CMS, as either *excellent*, *good*, *fair* or *poor*.

Functionality

1. Suitability: the presence and appropriateness of controls for different tasks. For ex. if you wanted to backup site content, was there an appropriate control (button/link) present?
2. Accurateness: how accurate were the results of system's tasks.
3. Functions evaluation: add page, edit page etc.

Usability

1. Understandability & learnability: how easy/hard was it to understand and then learn the system.
2. Operability: the ability of the interface to allow the control of the software. For ex. if you wanted to change an aspect of the software (such as language, style of the admin interface etc.) was the interface able to do that?
3. Appropriateness of menu, toolbar, buttons and the user interface as a whole.

Reliability

1. Faults & errors you came across. Did you come across many errors?
2. How satisfied were you with error handling. If you encountered an error (e.g. page wasn't saved as it should have been) were you satisfied with the error message provided (not)?
3. Consistency of system performance over a period of time.

Efficiency

1. Efficiency: the level of productivity once you have learned the system.
2. System resources used by the system (sluggish behavior etc.). For ex. you were trying to save a page and it took ages.

Comments about CMS

Like What do you like about the CMS?

Missing What needs to be improved?

Comments Anything else? 'Get off your chest'.

A.2 Results

Figure 14: System evaluation results.

	B	R	M1	M2	J	**Average**
Suitability	3	4	3	4	4	3.60
Accurateness	3	3	4	4	4	3.60
Functions	4	3	4	4	3	3.60
Understandability	4	2	4	4	4	3.60
Operability	2	3	3	4	4	3.20
Appropriateness	3	3	2	3	4	3.00
Errors	2	3	3	3	4	3.00
Error Handling	3	2	3	4	4	3.20
Consistency	4	3	4	4	4	3.80
Efficiency	4	3	4	4	4	3.80
Resources	4	4	4	4	4	4.00

A.3 Comments

Not exhaustive, translated from Czech.

The Good

- Copying text from other files is very effective and its editing is easy in the system.
- I believe that in today's age (the system) will work for a very wide layman clientele on my "IT level".
- This is the first time in my file that I feel like a "small web designer", I don't have to rely on professional services of others and can continuously update my web.

- I like the easy navigation in the menu, well-arranged adding and editing of items but also an easy management of uploaded files and working with back-ups.
- Kreus CMS is an application that will satisfy users looking for simplicity and administrators that need to setup their systems such that it's well-arranged for such users. That's also why I rate Kreus CMS as a well-made system that satisfies premises laid out for itself.
- I like the backup function that automatically generates date and time.

The Bad

- I cannot edit line width, can't move one-word syllables onto a next line (which is grammatically incorrect in Czech).
- I would like to see a preview of the page I am editing.
- I can't upload a big image.
- I would add more settings and better user management privileges.
- Edit and Delete Page function icons are too close together, I might click on the wrong one.
- Can I have arrows in the heading of a table to sort it?
- If I want to add an image to the page I need to know its web address. But my file is on a hard disk and I need to upload it first - not easy and intuitive.

B XHTML Prototypes

Figure 15: First XHTML Prototype.

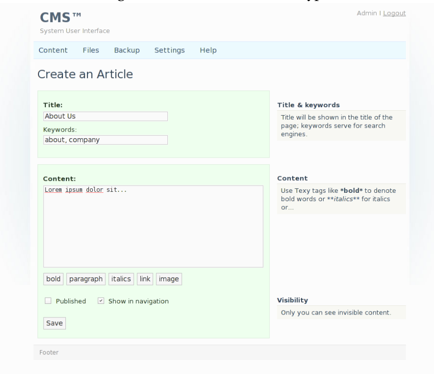


Figure 16: Second XHTML Prototype.

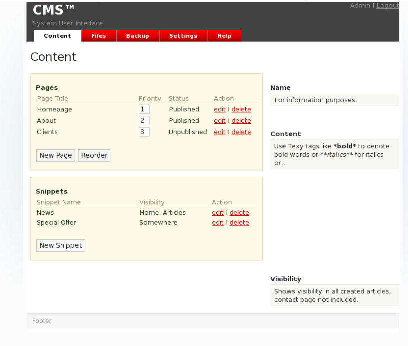


Figure 17: Third XHTML Prototype.

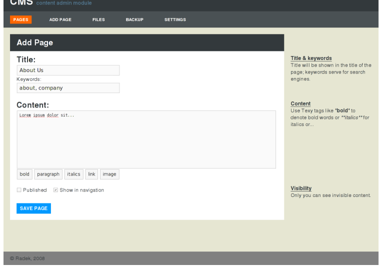
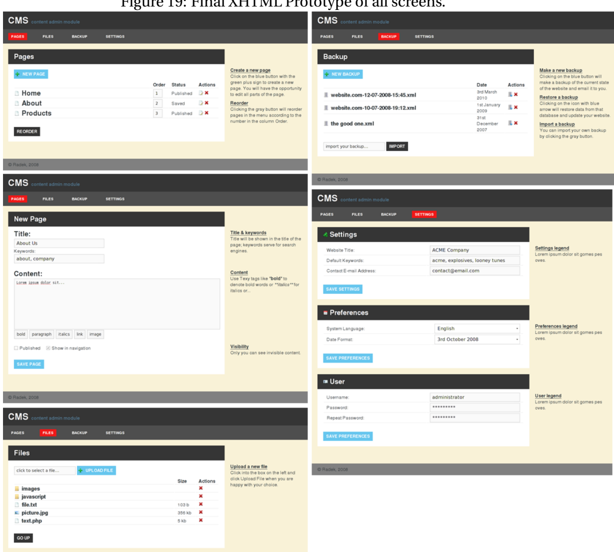


Figure 18: Fourth XHTML Prototype.

Figure 19: Final XHTML Prototype of all screens.



C Kreus CMS Admin Interface

Figure 20: Kreus CMS login screen.

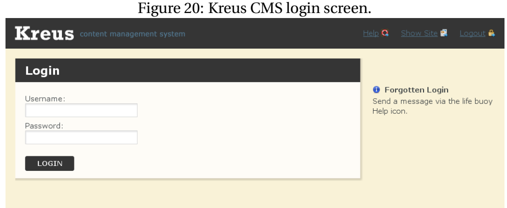


Figure 21: Kreus CMS pages screen.

Kreus
content management system
Help
Show Site
Logout
PAGES
FILES
BACKUP
SETTINGS
Pages
NEW PAGE

	Menu	Visible	Actions
Home	1.		
Documentation	2.		
Templating	0.		
Installation	0.		
Portfolio	4.		
Updates	3.		
Coding Standard	0.		

UPDATE STATUS

Create a new page
Click on the dark 'New Page' button to create a new page.
Homepage
The image of the house next to a page name shows you which page is currently set as the homepage.
Menu
The number represents the position of that page in the main website menu. A zero means the page won't be shown in main menu.

Figure 22: Kreus CMS edit page screen.

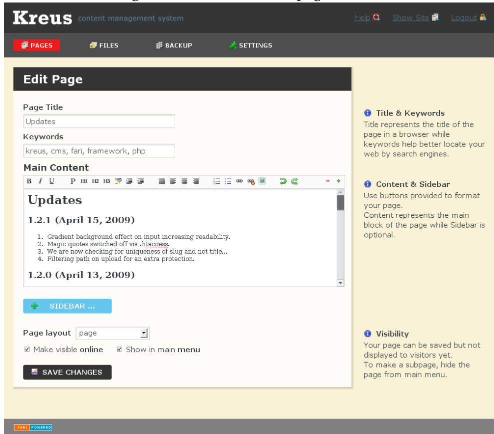


Figure 23: Kreus CMS files screen.

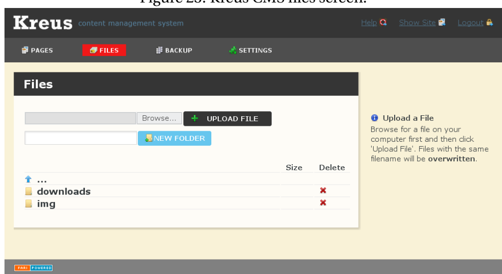


Figure 24: Kreus CMS backup screen.

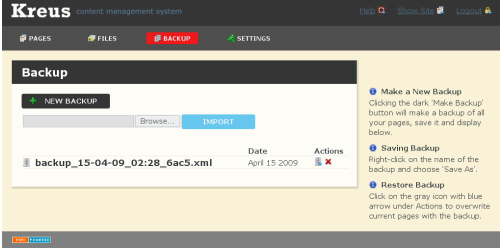
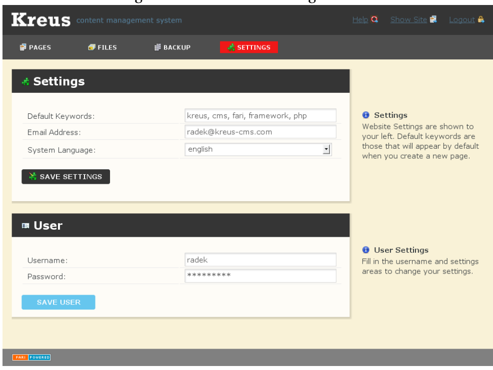


Figure 25: Kreus CMS settings screen.



D Kreus CMS Browser Compatibility

Figure 26: Kreus CMS under Mozilla Firefox 3.

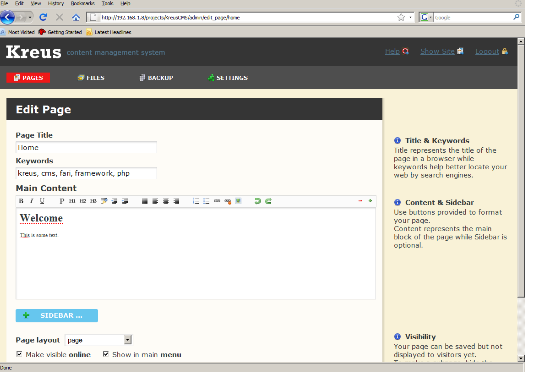


Figure 27: Kreus CMS under Opera.

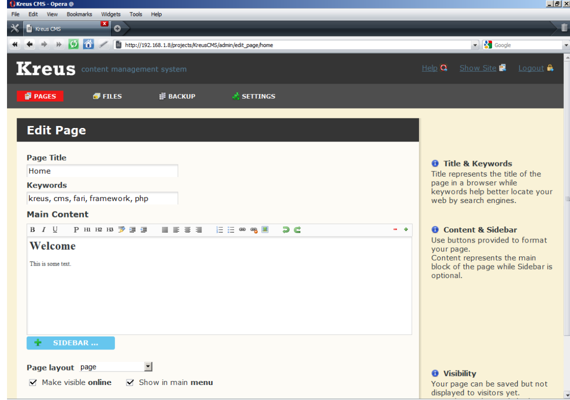


Figure 28: Kreus CMS under Safari.

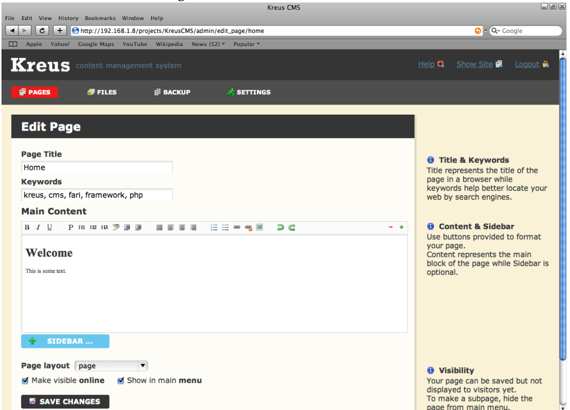


Figure 29: Kreus CMS under Chrome.

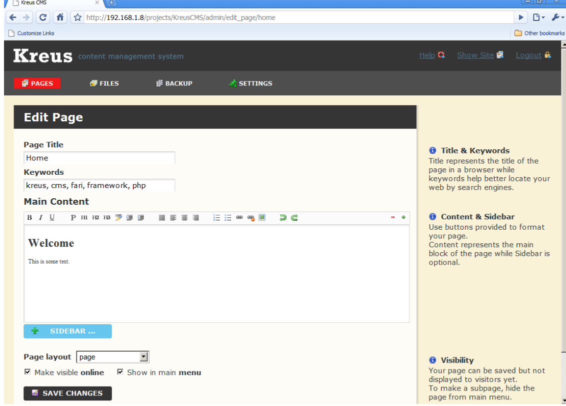
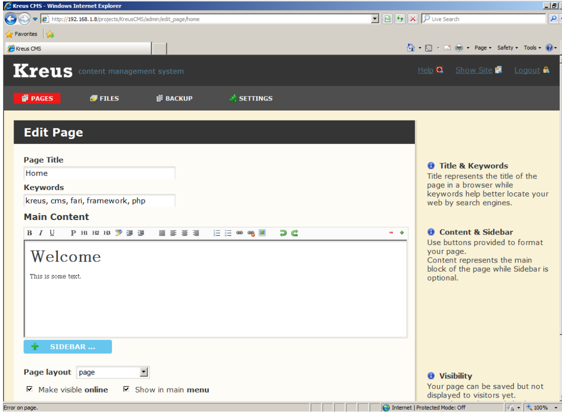


Figure 30: Kreus CMS under Internet Explorer 8.



E XSS

Figure 31: Acunetix Web Vulnerability Scanner XSS scan result.

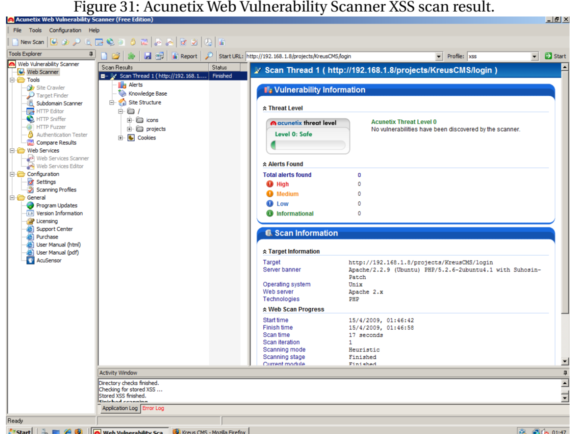
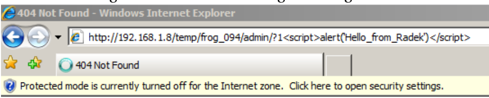
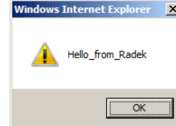


Figure 32: Successful XSS against Frog CMS.





F Fari MVC Framework Screenshots

Figure 33: Fari Framework handling an exception of a missing view.

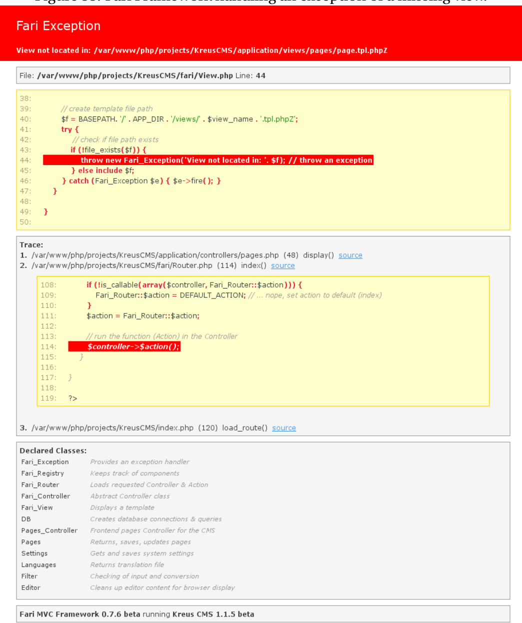
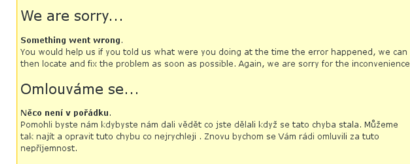


Figure 34: Fari Framework handling an exception on a production server.



G Fari MVC Framework Q&A

Why doesn't the framework provide some sort of an admin interface like Django, Symfony etc?

To not encourage the traditional way of developing an application, that is, coding first, design second. Instead it forces developers (to an extent) to do the Views first and then code the hard part (Models).

Why don't models extend a parent model class?

The idea is to allow developers to reuse their existing code easily. This is best done if the only thing needed to be done when creating a model is to save it into an appropriate directory, such as is now.

Why aren't more models included with the standard distribution of Fari MVC?

Models from Kreus CMS can be reused but often times they will have to be edited and tested heavily [FactsEng03, ch.1] to serve the needs of that particular developer.

What are all those *class_text()* functions?

Fari_Exception handler checks for these and if they are implemented, will show next to the declared class name in case of an exception being thrown. This way you don't see a long list of class names and/or file names but rather descriptions of what these actually do.

Is there a way to authenticate users easily?

Yes, if you implement an *authorize_user()*[49] function, it is called when a Controller is constructed, before any calls to Actions (pages) are made. You set it and forget it knowing that any function implemented in the Controller has the same protection.

Why are there so many updates released for the framework and CMS?

Released doesn't mean without error or perfectly fine tuned as it is nearly impossible to test software at the level of 100% of its logic paths [FactsEng03, ch.2]. As the time spent with the project in real life increases, so does the number of ideas for a better product increase. The code is continuously done bit by bit and is partially executable every day. This is what is called simplicity: gracefully solving today's problem [AgilePract08, p.63].

[49] Or any other function as specified in the Front Controller *index.php* file.

H Project Management & Updates

H.1 Preparation & Setup

Figure 35: Preparation & Setup stage.

TERM	WEEK	DATE	ACTIVITY	NOTES
AUTUMN	1	10th Oct 2008	Preparation	Goplan.org picked as a project management tool
				Need to read about Ajax and MVC
				Decision to use Textile to edit content
		11th Oct 2008		
		12th Oct 2008		Getting Real by 37signals and Agile philosophy adopted
	2	13th Oct 2008		URI and Ajax ideas
		16th Oct 2008		
		18th Oct 2008	Setup	Decide on Requirements for the system
				Git picked as a version control tool
				Apache, MySQL install
		19th Oct 2008		Shell script to backup my work on external drive
	3	22nd Oct 2008	Project Proposal	

H.2 XHTML Prototypes

Figure 36: XHTML Prototyping stage.

TERM	WEEK	DATE	ACTIVITY	NOTES
AUTUMN	3	23rd Oct 2008		First CMS pencil on paper mockup
		24th Oct 2008		Content split decision made, more mockups
		25th Oct 2008		First XHTML mockup
		26th Oct 2008		Second XHTML mockup
	4	28th Oct 2008	XHTML	Angry with the design direction
		29th Oct 2008		Third XHTML mockup
		30th Oct 2008		
	5	3rd Nov 2008		Fourth XHTML mockup
		4th Nov 2008		
		5th Nov 2008		Final XHTML mockup of all screens
		6th Nov 2008		
	6	11th Nov 2008		
		15th Nov 2008		
	7	18th Nov 2008		Finished usability testing with my 2nd person

H.3 Model-View-Controller Framework

Figure 37: MVC Framework development stage.

TERM	WEEK	DATE	ACTIVITY	NOTES
AUTUMN	7	19th Nov 2008	MVC Framework	Reading Nettuts, Nemetral, PaperBagCoder articles about MVC
		20th Nov 2008		Car.mdl MVC Framework
				A gentle introduction to MVC
		21st Nov 2008		Discussing MVC ideas with Nemetral
		22nd Nov 2008		Building MVC
		23rd Nov 2008		Main work on Fari MVC is finished
	8	24th Nov 2008		Benchmark class in the libraries
				Simple insert, select and update queries in DB model
		25th Nov 2008		
		26th Nov 2008		A demo application under Fari MVC
		28th Nov 2008		Fari MVC has a custom Exception thrower
		29th Nov 2008		Framework split into /app /fari and /public folders
				Custom error route in Router class
				Router class split into more functions, reused
				A demo Blog application
				DB model now has prepared statements, queries can be reused
				File handling model
		30th Nov 2008		More extensive users model, password SHA-1 encrypted

H.4 Content Management System Development

Figure 38: CMS development stage.

TERM	WEEK	DATE	ACTIVITY	NOTES
AUTUMN	9	2nd Dec 2008	CMS	Controllers for CMS
		3rd Dec 2008		XHTML Prototype screens saved as .tpl.php Views
		4th Dec 2008		Plugging in Models
		5th Dec 2008		Dynamic Pages loading functionality
		6th Dec 2008		Pages, Files, Settings done
				Crude implementation, basic functionality
	10	9th Dec 2008	Interim Report	
		10th Dec 2008		
		14th Dec 2008	Textile	Basic markup to XHTML converting
BREAK	1	18th Dec 2008		Unordered lists now added to Textile class
SPRING	3	26th Jan 2009	CMS	Backup Import/Export
		27th Jan 2009		Settings done
				Updating username & password on Setting page
		28th Jan 2009		Template folders restructured to make the app more easily readable
		29th Jan 2009		Class documentation extended
				Ajax helper functions in Admin controller
		30th Jan 2009		Sitemap and Contact Form served via Pages controller
		31st Jan 2009		Public upload folder no longer hardcoded, file stats (size, type) returned in listing
		1st Feb 2009		Email, slug, number, alphanum checking in Filter model
				Functions returning page content for menu, listing, page; reuse
				Kreus CMS 1.0.0 released
	4	3rd Feb 2009	Fixes, updates	
		4th Feb 2009		A separate Settings class used for easy update of only the system
				Work on file manager and permissions now chmoded
				Page update SQL queries split into separate instances with string escape
				Session user id is encoded via SHA-1
		5th Feb 2009		Fix file manager error, deleting a file within a directory
				SMTP via PEAR::Mail added to the Forms model
		6th Feb 2009		Textile class functionality extended, , <p, <img>, <a> tags
		7th Feb 2009		Correct kilobits size returned in files model
				Plugin model added that allows for external markup language to parse
				Texy! Markup language added as an external plugin
				Kreus defaults to Textile if plugin not found
		8th Feb 2009		Icon in the footer showing external plugin use
				Slashes stripping function if magic_quotes on in filter model
				Single quotes escaped by converting to HTML char
				Exhaustive regex engine documentation
	5	13th Feb 2009	Dissertation	
		14th Feb 2009		
		15th Feb 2009		
	6	16th Feb 2009		
		17th Feb 2009		
		18th Feb 2009	Fixes, updates	Kreus CMS version released under MIT License

H.5 Further CMS Development

Figure 39: Further CMS development stage.

TERM	WEEK	DATE	ACTIVITY	NOTES
SPRING	6	19th Feb 2009	Fixes, updates	Fixed glitch in Database class where only localhost server was resolved
		20th Feb 2009		SMTP via PEAR::Mail or PHP mail() used based on which settings are provided
		22th Feb 2009		Email head provided for mail() to prevent messages being labeled as spam
	7	23rd Feb 2009		Logger of actions added so I can tell if test users are using the system
		27th Feb 2009		Diacritics stripped when creating a page slug from the page title
	10	21st Mar 2009		Password updated first to prevent bug when changing username as well
				Backup directory and files obscured
				Major security flaw fixed in user authentication, forgot to die()
BREAK	1	24th Mar 2009		Newer version of Mootools used, Ajax Requests had to be updated
				Textarea can be now resized
		26th Mar 2009		Input escaping and encoding sorted
		27th Mar 2009		Work on Rich Text Editor implementation
		28th Mar 2009	Rich Text Editor	Elmo Rich Text Editor released
		29th Mar 2009		Major overhaul of the system
				Elmo Rich Text Editor used instead of a classic textarea, no need for Texy or Textile
	2	1st Apr 2009	Dissertation	Project management notes and updates put together
		2nd Apr 2009		Project management added to the Appendix
		3rd Apr 2009		Writing about Elmo Rich Text Editor
		5th Apr 2009	Framework	Fari MVC 0.6.0 fine tuned with less code and more comments
	3	6th Apr 2009		Fari MVC 0.7.0 now has a 'beautiful' Exception class, ala Nette Framework
			CMS	Elmo RTE now trims input off whitespace chars with a custom function
		7th Apr 2009		I18n folder created so only the required translations are loaded
			Framework	Exception thrower shows more highlighting and a trace
		9th Apr 2009		Declared classes and their descriptions implemented
			CMS	Page title and slug updates, glitch of not checking for unique title in edit page fix
		11th Apr 2009		Better filename checking on the side of file uploader
				SHA-1 encrypted user id checked with the database for match, more secure
				A separate Controller for the login process used
			Framework	Cleaned up route requested passed as a parameter to every Controller to be reused
				authorize_user() called during Controller construct checking id, faster, better
	4	13th Apr 2009		A more user friendly error message shown on production server
			CMS	A custom layout can be selected for a page, all automatic from Views folder
				RSS Feed template added
				Backup glitches fixed; Block formatting in Editor; increased readability of Edit Page
		15th Apr 2009		Background effect on input for increased readability
				Magic quotes switched off via .htaccess to standardize distros
SUMMER	1	20th Apr 2009	Dissertation	Final polish applied

I Kreus CMS Source Code

- The attached medium contains Kreus CMS ver. 1.2.1, released April 15 2009, in the file KREUSCMS_1.2.1.ZIP.
- The project's homepage is `http://kreus-cms.com`.