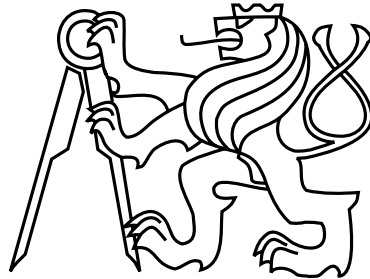


Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering



Master's Thesis

Web content management system: architecture, concepts and application

Bc. Josef Kunhart

Supervisor: Ing. Miroslav Bureš, Ph.D.

Study Programme: Electrical Engineering and Information Technology

Field of Study: Computer Science and Engineering

April 28, 2013

Aknowledgements

I would like to thank my master's thesis supervisor, Ing. Miroslav Bureš, Ph.D., my relatives and friends for continual support and patience during writing this work.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.

I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000 Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague on April 27, 2013

.....

Abstrakt

Cílem diplomové práce je návrh webového systému pro správu obsahu. Důraz je kladen na architekturu, návrh databáze, dílčí koncepty a uživatelské rozhraní systému. V textu jsou popsány typicky řešené dílčí koncepty a problémy webového systému pro správu obsahu. U klíčových konceptů jsou diskutovány různé varianty řešení. Tyto varianty jsou porovnávány s podobnými řešeními a produkty. V první části práce je popsána architektura systému, v dalších částech důležité koncepty a poslední část se zabývá návrhem uživatelského rozhraní a uživatelským testováním. Jednotlivá témata jsou diskutována v rovině návrhu s přesahem do implementace, detailní popis implementace programu není cílem této práce.

Abstract

The aim of this study is to propose design of a web content management system. This work focuses on architecture, database design, local concepts, and user interface design. Key concepts and typical problems of a content management system are described. Multiple solutions for key concepts are investigated and compared to similar solutions and products. The first part of the study explains system architecture, the following parts discuss global and local concepts, and the last part deals with the user interface design and user testing. This work focuses on the application design without a detailed description of the program's implementation.

Contents

1	Introduction	1
1.1	State of the Art	1
1.2	Goals of the Study	1
1.3	Project Name	2
1.4	Urchin Overview	2
1.5	Urchin Use	2
1.6	Technology Used	3
1.7	Thesis Outline	3
	1.7.1 Introduction	3
	1.7.2 Main Content	3
	1.7.3 Conclusion	4
2	Architecture	5
2.1	Chapter Overview	5
2.2	Requirements	5
	2.2.1 Functional Requirements	5
	2.2.2 Non-functional Requirements	6
	2.2.3 Software Requirements	7
2.3	Architecture Design	7
	2.3.1 Model-View-Controller Architectural Pattern	7
	2.3.2 Hierarchical Model-View-Controller Architectural Pattern	8
	2.3.3 Front Controller Design Pattern	9
	2.3.4 Use of Hierarchical Model-View-Controller in Urchin CMS	9
2.4	Packages	10
2.5	Context and Core Objects	11
	2.5.1 Context	11
	2.5.2 Session	11
	2.5.3 Request and Response	11
	2.5.4 Pool	12
	2.5.5 Cache	12
	2.5.6 Link	12
2.6	Database Design	12
	2.6.1 Database Layer	12
	2.6.2 Database Structure	13
	2.6.3 Alternative Approaches	13

2.7	Crud Library	13
2.7.1	Common and Element Crud	15
2.7.1.1	Entries	17
2.7.1.2	Components	17
2.7.1.3	Decorators	17
2.7.1.4	Validators	17
2.7.1.5	Converters	17
2.7.1.6	Filters	18
2.7.2	Cross and Matrix Crud	18
2.7.3	Flexibility and Extensibility	18
3	Core Features and Modules	19
3.1	Chapter Overview	19
3.2	The Page Axis	19
3.2.1	Website	19
3.2.2	Presentations	20
3.2.3	Pages	20
3.2.4	Templates and Positions	21
3.2.5	Page Parameters and System Pages	22
3.3	Internationalization and Localization	22
3.3.1	File-Based Translations	23
3.3.2	Database-Based Translations	23
3.4	The Component Axis	23
3.4.1	Modules	23
3.4.2	System Modules	24
3.4.3	Content Modules	25
3.4.3.1	Application Modules	25
3.4.3.2	Element Modules	25
3.4.4	Components	26
3.4.5	Elements	27
3.4.6	Module Templates and Parameters	28
3.4.7	Linkable Modules	28
3.5	Security of a Web Application	28
3.5.1	SQL Injection	28
3.5.2	Cross-Site Scripting	29
3.5.3	Authentication and Session Management	30
3.5.4	Cross-Site Request Forgery	30
3.6	The Permission System	31
3.6.1	Authentication and Authorization	31
3.6.2	General Access Control	31
3.6.3	Action and Data Permission	32
3.6.4	User and Groups	32
3.6.5	Permission for Modules and Actions	33
3.6.6	Permission for Presentations	34
3.6.7	Permission for Page Groups	35
3.6.8	Additional Features	35

3.6.9	Front-End Permission	36
3.7	Additional Features	36
3.7.1	Content Work-Flow	36
3.7.2	Content Preview	37
3.7.3	Personal Settings	38
4	Extending Modules	39
4.1	Chapter Overview	39
4.2	Content Modules	39
4.2.1	Articles	40
4.2.2	Content	40
4.2.3	Enquiries	40
4.2.4	Events	40
4.2.5	Forms	41
4.2.6	Galleries	41
4.2.7	News	41
4.2.8	QuickContact	41
4.2.9	RSS	41
4.2.10	Search	41
4.2.11	Sitemap	42
4.3	Dynamic Forms	42
4.3.1	Form Implementation	42
4.3.2	The Form Library	43
4.3.3	Controls	43
4.3.4	Validators	43
4.3.5	Form Processing	44
4.3.6	Form Security	44
4.3.7	QuickContact Revisited	45
4.3.8	Forms Revisited	45
4.4	Content Search	46
4.4.1	Entity-Based Search	46
4.4.2	Content Indexing	47
4.4.3	Combined Search	49
4.4.4	Search Engine Service	49
4.4.5	External Search Engine	50
4.4.6	Index Module	50
4.4.7	Search Module	50
4.4.8	Indexing Table	51
5	User Interface Design	53
5.1	Administration	53
5.1.1	Login Screen	53
5.1.2	Administration Layout	54
5.1.3	Page View	56
5.1.4	File View	58
5.2	Crud Layout	58

5.3	Front-end Layout	60
6	Testing of User Interface Design	63
6.1	Objective	63
6.2	Target Group	63
6.3	Test Participants	64
6.3.1	User A	64
6.3.2	User B	64
6.4	Screener	64
6.4.1	Questions	64
6.4.2	Selection of Participants	65
6.4.3	Screener with User A	65
6.4.4	Screener with User B	65
6.5	Interview	65
6.5.1	Topics and Questions	65
6.5.2	Interview with User A	66
6.5.3	Interview with User B	67
6.5.4	Summary of Information	68
6.6	User Scenarios	69
6.7	Low Fidelity Testing	70
6.7.1	Testing Overview	70
6.7.2	Paper Prototype	71
6.7.3	Goals and Scenarios	71
6.7.4	Testing Plan	72
6.7.5	Testing with User B	72
6.8	Testing with User A	73
6.8.1	Testing Summary	75
6.9	High Fidelity Testing	76
6.9.1	Testing overview	76
6.9.2	Goals and Scenarios	76
6.9.3	Preparations	76
6.9.4	Testing Plan	76
6.9.5	Testing with User B	77
6.9.6	Testing with User A	78
6.9.7	Evaluation of Testing	78
7	Conclusion	81
7.1	Achieved Objectives	81
7.2	Future Plans	81
7.2.1	Content Versions	81
7.2.2	File Management	81
7.2.3	Client Zone	82
7.2.4	Technical Improvements	82
7.3	New Modules	82
7.4	Licensing	83

A	List of Abbreviations	89
B	List of Terms	91
B.1	Architecture	91
B.2	Core Features and Modules	92
B.3	Extending Modules	93
B.4	User Interface and Testing	93
C	List of Content Modules	95
C.1	Basic Modules	95
C.2	New Modules	96
D	Electronic Disc	97
E	Database Model	99
E.1	Core Application	99
E.2	Content Modules	99
F	Sample Application	101
F.1	Description	101
F.2	Website and Presentations	101
F.3	Templates and Positions	101
F.4	Hierarchy of Pages	102
F.5	Modules and Components	102

List of Figures

2.1	Schema of modified MVC pattern with router and user interaction.	8
2.2	Schema of hierarchical MVC pattern.	8
2.3	Hierarchical MVC pattern applied to Urchin CMS.	9
2.4	Urchin CMS structure at the package level.	10
2.5	Database model with core tables and their relations.	14
2.6	Packages and classes of the crud library.	15
2.7	State diagram for common crud with transitions between actions.	16
3.1	Schema of the page axis with presentation, pages and templates.	20
3.2	A sample website with two presentations and several top-level pages.	21
3.3	Schema of the component axis with modules, components and elements.	24
3.4	Organization of content modules into element and application sub-modules.	26
3.5	The relation between the element table and module-specific tables.	27
3.6	Login process in a regular web application with authentication and authentication.	32
3.7	Database tables of the permission system and their relations.	33
3.8	Managing access rights to actions in the detail of the user group.	35
3.9	The life cycle of an element with visible and hidden work-flow states.	37
4.1	Simple contact form with three mandatory fields: subject, e-mail, and message.	42
4.2	Form processing diagram with views and actions.	44
4.3	Schema of the entity-based search method.	47
4.4	Schema of the context indexing search method.	48
4.5	Schema of the combined search method.	49
4.6	Sequence diagram for the indexing process.	51
4.7	Sequence diagram for the search process.	52
5.1	Login screen for the administration.	53
5.2	Basic layout and main panels in the administration.	54
5.3	Logical organization of actions in the page view.	55
5.4	Page view with hierarchy of pages and page detail.	57
5.5	File view with hierarchy of directories and directory detail.	57
5.6	The view action of an element crud instance showing sample data.	59
5.7	The edit action of an element crud instance displaying a sample record.	59
5.8	The view action of a cross crud instance nested inside a common crud instance.	60
5.9	Basic layout of a front-end page.	61

6.1	Paper mock-up overview with different types of paper components.	71
6.2	User B added a new page for articles.	73
6.3	User A after adding a new component.	74
6.4	Place prepared for the high fidelity testing.	77
F.1	Templates available in the sample application.	102
F.2	Logical hierarchy of the sample website.	103

List of Tables

3.1	Default user groups available in the Urchin application.	33
3.2	Options available to the modules.	34
3.3	Sample modules of various types with assigned options.	34
3.4	Front-end user groups available in the Urchin application.	36
3.5	Summary of work-flow states and their use.	37
4.1	Overview of basic content modules.	40
4.2	Standard controls and validators used in the QuickContact module.	45
4.3	Form fields available in the Form module.	46
5.1	Sections in the administration with description and modules.	55
F.1	Templates and settings of pages in the English presentation.	103
F.2	Components assigned to pages of the English presentation.	104

Chapter 1

Introduction

1.1 State of the Art

In the beginning of 1990s, the web was nothing like in these days. Developers used simple text editor to build static pages. Users browsed web pages using a historical browser, such as Mosaic [27]. Most non-technically oriented people did not even know there is something called the web. Times have changed and now the Internet is much more important than ever before. Web technologies are accessible, cheap and widely used. Many companies now prefer online solutions to former desktop applications. Common users have realized that they have a lot of content to present and share. Today, the Internet serves as a great tool for sharing different content between users. Users take as granted web presentations and portals, online shopping and banking, full-text search, or social networks.

With the rise of the Internet, many companies and individuals have begun to present and propagate themselves on the web. Traditionally, a programmer or an internet agency was required to create and maintain a website. Managing a website without knowledge of at least HTML¹ [16] and CSS² [8] technologies was nearly impossible. This approach is ineffective, time-consuming and expensive for the company. Later, in the end of 1990s, a first CMS³ application emerged [47]. A CMS system is a type of web-based software that simplifies management of a website. This system allows easy content editing, templates and automated indexing of content for search. More advanced features of a CMS are e.g. specific modules, content preview, user roles and permission.

1.2 Goals of the Study

The first and the most important goal of this study is to propose design of a new content management system. Important fields such as architecture, database design, various concepts and user interface design are discussed. Proposed concepts and features are demonstrated on author's genuine solution, a content management system called Urchin CMS. This project has already started in January 2011 and is still being continuously developed and improved.

¹HyperText Mark-up Language

²Cascading Style Sheets

³Content Management System

The system serves as both an experimental platform and a valuable tool for website development. The limiting factor for development is time, as author both has a full-time job and studies at the university.

Secondary goal of this study is to discuss multiple ideas for key concepts. This contains comparison of the most important concepts with other approaches and existing solutions, e.g. massively used open-source content management systems, such as Drupal [10] or Joomla [24]. Pros and cons of these systems are mentioned as one of the reasons leading to the development of Urchin CMS. This study could also serve as a rough guide for architects or developers of content management systems. This work covers all important areas of the design and architecture.

1.3 Project Name

Before discussing project's architecture and features, three important names connected to the project should be clarified. This project began as a simple group of classes repeating in projects without any name or title. Later, a first version of framework emerged directly from these classes and has been assigned a working name Web Component Framework. After that, development of a new content management system has started. This system was built upon the WCF framework and assigned a working title Urchin CMS (or just Urchin). During the development of Urchin CMS version 2.0, the underlying framework was revised, simplified and then renamed to Web Component Framework: Simple.

1.4 Urchin Overview

Urchin CMS is a multi-purpose modular content management system. The system supports multiple presentations and languages, multiple users with different roles and advanced content handling. Advanced concept of components and elements provides great flexibility and effective management of the website content. Components are shared between presentations, there is no need to create and maintain duplicate content. Permission management for pages, modules and actions allows different access rights for each user. This is particularly useful in connection with built-in work-flow feature for approving content. Urchin CMS is a reliable, fast, and secure solution for small- to medium-size websites.

Urchin CMS installation comes with many basic modules, such as simple content, articles, news and forms. In example, dynamic form module is used to create different forms without having any programming knowledge. In addition, many new modules are planned for future development, e.g. forum, maps, products, or client section. The system is highly modular and well prepared for adding custom modules. Custom modules extend functionality according to client's specific requirements and needs.

1.5 Urchin Use

There are two main areas where Urchin CMS is expected to be used. The first area is web presentations. A web presentation features creative design and is usually focused on

offering services and presenting references. The content is static with addition of few forms. Even large web presentations contain hundreds of articles but only few dynamic features (forms, comments, enquiries). Advanced functions and custom modules are usually not used. A good example of a web presentation is a small company website, personal portfolio or product micro-site.

The second area is web applications. A web application is much more dynamic and functionally oriented than a common web presentation. Visual design and user interface of a web application is important as well as functions and content the application offers. Each application of this type is customized to client's needs and business logic using specialized modules. A typical web application is in example a CRM⁴ system, an e-shop, an educational system or an intranet.

1.6 Technology Used

From the technological point of view, Urchin CMS is a web-based client-server application. This application is written in PHP⁵ [35] 5.3 programming language and uses either MySQL [28] or PostgreSQL [40] database for storing its data. The presentation layer of the application is written using XHTML [50] and CSS⁶ [8] for formatting and jQuery [52] library for client-side scripting. Urchin CMS is divided into two main sub-applications: administration and front-end. Administration is used for managing website while front-end presents website content to visitors. User uses any modern web browser to access or manage website content. Technological requirements will be discussed in detail in the next chapter entitled [Architecture](#).

1.7 Thesis Outline

This thesis is divided into three main parts with eight different chapters.

1.7.1 Introduction

The first chapter [Introduction](#) serves as introduction to the project. This chapter defines the state of the art, study goals, and presents project overview.

1.7.2 Main Content

The following five chapters form the main body of the study. These chapters are most important and most comprehensive. Chapter [Architecture](#) describes project requirements, system architecture, important packages, and database design. Chapter [Core Features and Modules](#) is about global concepts of the system, such as pages, components, or security and permission system. Chapter [Extending Modules](#) describes modules and features that are used to manage website content of different types. Chapter [User Interface Design](#) discusses

⁴Customer Relationship Management

⁵PHP: Hypertext Preprocessor

⁶Cascading Style Sheets

user interface design of the system and chapter [Testing of User Interface Design](#) describes testing of user interface arranged to improve user experience with the system's administration.

1.7.3 Conclusion

The last chapter entitled [Conclusion](#) discusses the project future and encloses this study. It introduces new ideas and features for future versions and shortly evaluates the whole project and discusses achievements, drawbacks and possible improvements. Most of new features have already their place on the project roadmap and will be implemented in the future.

Chapter 2

Architecture

2.1 Chapter Overview

The second chapter is entitled Architecture and deals with global concepts, design, and architecture of a content management system. In this work, a global concept means an architectural aspect or a feature that has huge impact on application's design and architecture (in contrast to the extending modules that affect only minor areas). The initial section of this chapter state actual project requirements. The following sections describe application design, internal mechanics, crud library and security of Urchin CMS and web applications. The design part describes system architecture, core objects and database layer. Sections about internal mechanics describe working with pages, modules, and website content. The last part describes both general security of web application and permission system of Urchin CMS.

2.2 Requirements

This short section serves as an introduction to further topics in this chapter and clearly sums up all requirements for Urchin project. Project requirements reflect contemporary expectations for the system as well as historical development. All requirements stand for Urchin CMS up to the version 2.2. This iteration is planned for the year 2013, as noted before. Only fundamental requirements are listed, otherwise the list would be too complex and confusing.

2.2.1 Functional Requirements

Functional requirements describe functions, processes and behaviour of the application. In other words, these requirements represent what the system is required to do. All requirements are sorted by their priority. The first list shows application-wide functional requirements:

1. website management - presentations with tree-structured pages
2. content management - system of components, multiple content on each page
3. multi-language system - each presentation might use different language or locale
4. multi-user access - multiple users could use administration at the same moment
5. permission management - user groups and accounts, system of access rights
6. crud library - tool for generation of basic user interface
7. WYSIWYG editing - editing of content without advanced knowledge
8. online preview - user can preview edited content before its publishing
9. caching - reduction of database queries and load
10. content versioning - content revisions on the element level

The second part shows module functional requirements:

1. basic modules - basic content, news, articles, simple form
2. full-text search - search facility and indexing of content
3. advanced modules - dynamic form, gallery, products, linked modules
4. client section - front-end registration, login, and differentiated access

2.2.2 Non-functional Requirements

Non-functional requirements specify attributes and constraints of the application. In other words, these requirements are about expected qualities of the project, mostly from the technical point of view. Opposite to the previous list, requirements in this list are sorted alphabetically, not by priority. Most of these requirements are on a similar level of importance.

- flexibility - adding custom modules or diverse extensions is simple
- layered architecture - separated programming code, database queries and templates
- modern architecture - object-oriented approach, use of software engineering methods
- performance - programming code is effective, queries are optimized and cached
- simplicity - simple, minimalistic design with structured and documented code
- security - secure application according to actual requirements
- transactions - all database operations use transactions
- usability - well-designed, intuitive and comprehensive user interface

2.2.3 Software Requirements

Software requirements define software technology required to run Urchin CMS application. In most cases, a LAMP¹ stack is used for working web applications and WAMP² stack is used for development.

1. Unix/Linux/BSD - operational system for web server
2. Apache 2.2+ - web server
3. PHP 5.3+ - server-side programming language
4. MySQL 5.1+ with InnoDB [21] engine or PostgreSQL 8.4+ - database server
5. jQuery - client-side scripting language
6. XHTML 1.0 Strict, CSS 3.0 - templates and layout
7. modern browser, resolution 1280x1024 or more - administration access

2.3 Architecture Design

2.3.1 Model-View-Controller Architectural Pattern

Model-view-controller is one of the most influencing and commonly used software architectural patterns. This pattern was originally proposed in 1979 by Trygve M. H. Reenskaug [45], a Norwegian computer scientist. MVC pattern was proposed as solution for complex software systems working with large data sets and user interaction. This architectural pattern affects the entire application, unlike design patterns that provide solutions for more concrete problems of a software system. Design patterns used in this project will be mentioned in the end of this chapter.

Main purpose of the model-view-controller architectural pattern is separation of user interaction and data representation into three parts. The model part encapsulates application data and business rules. The controller part converts user actions to updates of model and changes in view. The view part displays output based upon information from the model. Multiple views might exist for each controller, e.g. a web template, a XML³ [51] file, or a PDF⁴ [34] document. Figure 2.1 presents modified version of this pattern. This variant is also called model-view-presenter [26]. This modification is typically used in web applications and uses a controller to access the view from the model instead of direct access. The router part in the diagram serves as a designated controller for routing requests to other controllers.

¹Linux Apache MySQL PHP

²Windows Apache MySQL PHP

³eXtensible Mark-up Language

⁴Portable Document Format

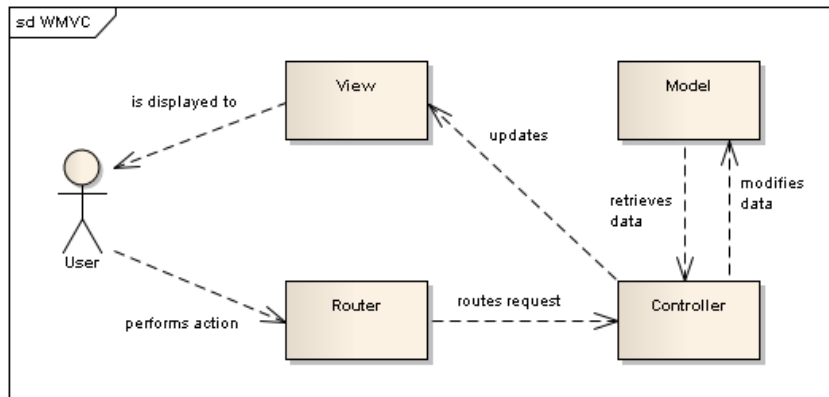


Figure 2.1: Schema of modified MVC pattern with router and user interaction.

2.3.2 Hierarchical Model-View-Controller Architectural Pattern

Hierarchical model-view-controller is an extended version of basic MVC pattern, inspired by an older presentation-abstraction-control architectural pattern [53]. For detailed information about all described architectural patterns, see [55]. As seen in figure 2.2, this pattern is composed of multiple MVC triplets arranged in a tree hierarchy. Each triplet is relatively independent and composed of all three parts, model, view, and controller. In addition to handling user actions, controller part is responsible for communication between the nodes. First, an action is redirected by the router to a controller in the root-level layer. After processing in this layer, controllers in the child layer are addressed.

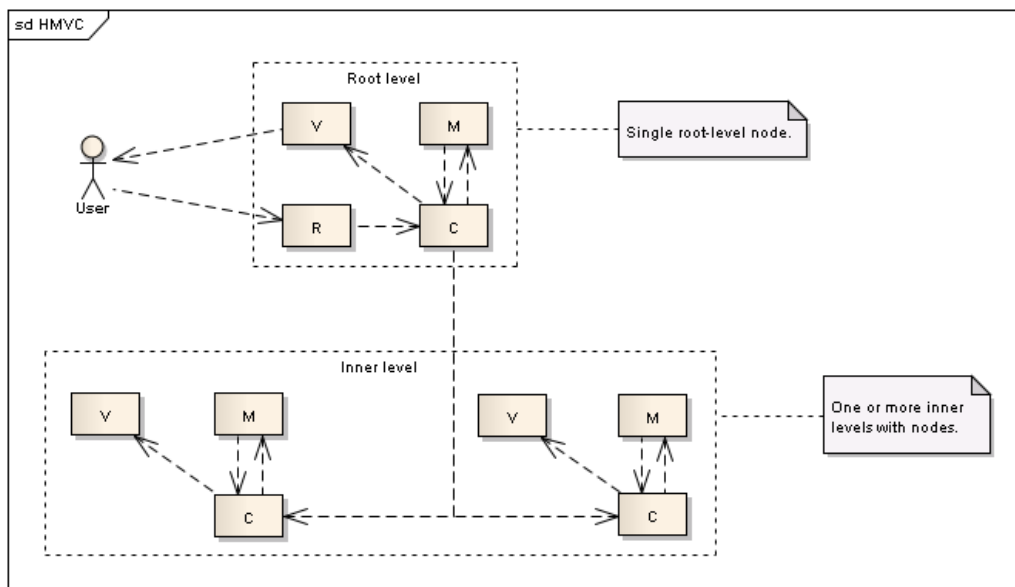


Figure 2.2: Schema of hierarchical MVC pattern.

2.3.3 Front Controller Design Pattern

Front controller [54] is a software design pattern, used especially in web applications. Author of the pattern is Martin Fowler, a well-known British software engineer. Main idea of this pattern is to provide a single point that receives and handles user requests for the whole web application. When used in cooperation with HMVC, the front controller serves also as the router part in the root-level of hierarchy. Usually, it is also responsible for security, checking permission, and localization.

2.3.4 Use of Hierarchical Model-View-Controller in Urchin CMS

As stated before in the introductory chapter, Urchin CMS is a web-based application. The application is divided into two major sub-applications, front-end and administration. Both parts are based upon HMVC architectural pattern and use two- or three-level deep hierarchy of nodes. The database layer represents model on all levels, the controller and the view parts differ. Details of use will be shortly described in the following text. Several features (e.g. database layer, modules, or crud library) presented in this text will be described in detail later in this or the following chapter.

In the front-end, each page has assigned a single template and is composed of multiple components. The root level of hierarchy applies to the whole front-end application and is composed of the web front controller, database layer, and a page layout. On the lower level, each node consists of a module controller and a module template. More complex modules, such as dynamic forms, use a third level that handles concrete form controls, e.g. input or radio control. See the left half of figure 2.3 for further details.

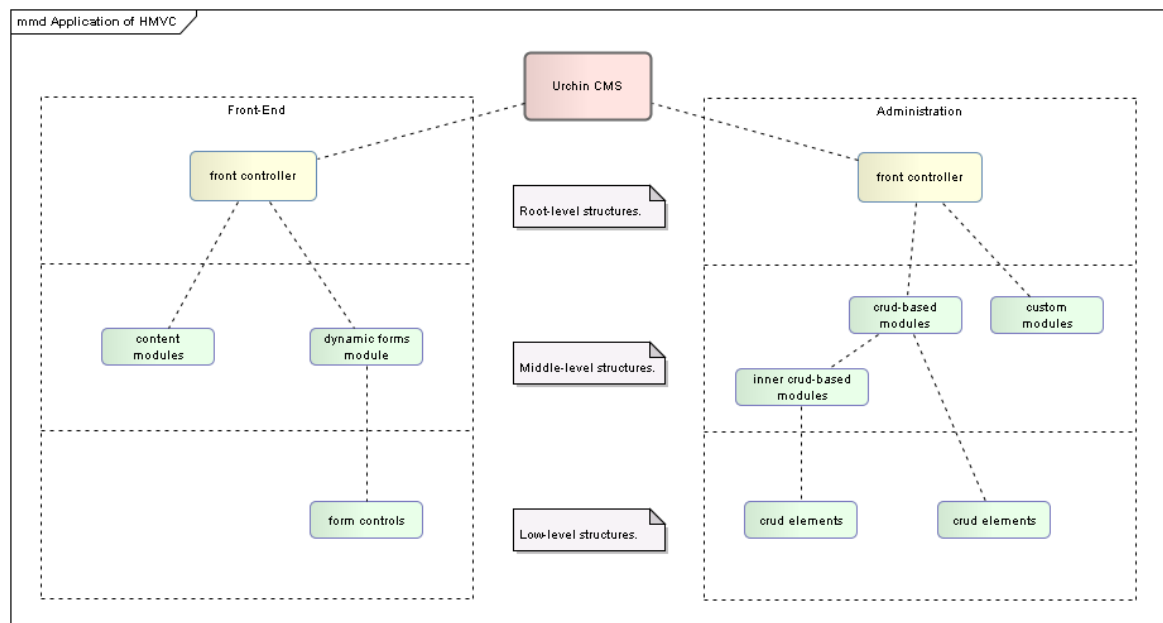


Figure 2.3: Hierarchical MVC pattern applied to Urchin CMS.

The architecture of the administration is similar to the front-end. The view part of the top level structure is represented again by the database layer, the administration front controller and administration layout. There are also two designated front controllers for handling cron and AJAX requests. On the middle level, each node is represented by a crud instance or a custom module. Every crud instance or custom module uses its own controller and templates. Crud instances might be nested and usually contain another level of hierarchy, represented by crud elements such as components and filters. See the right half of figure 2.3 for details.

2.4 Packages

Following the architecture design, Urchin application is divided into multiple packages and sub-packages. Each package is directly mapped to a PHP namespace and consists of number of classes stored in a separate directory. Urchin CMS is a completely object-oriented application and includes more than 300 classes of different types.

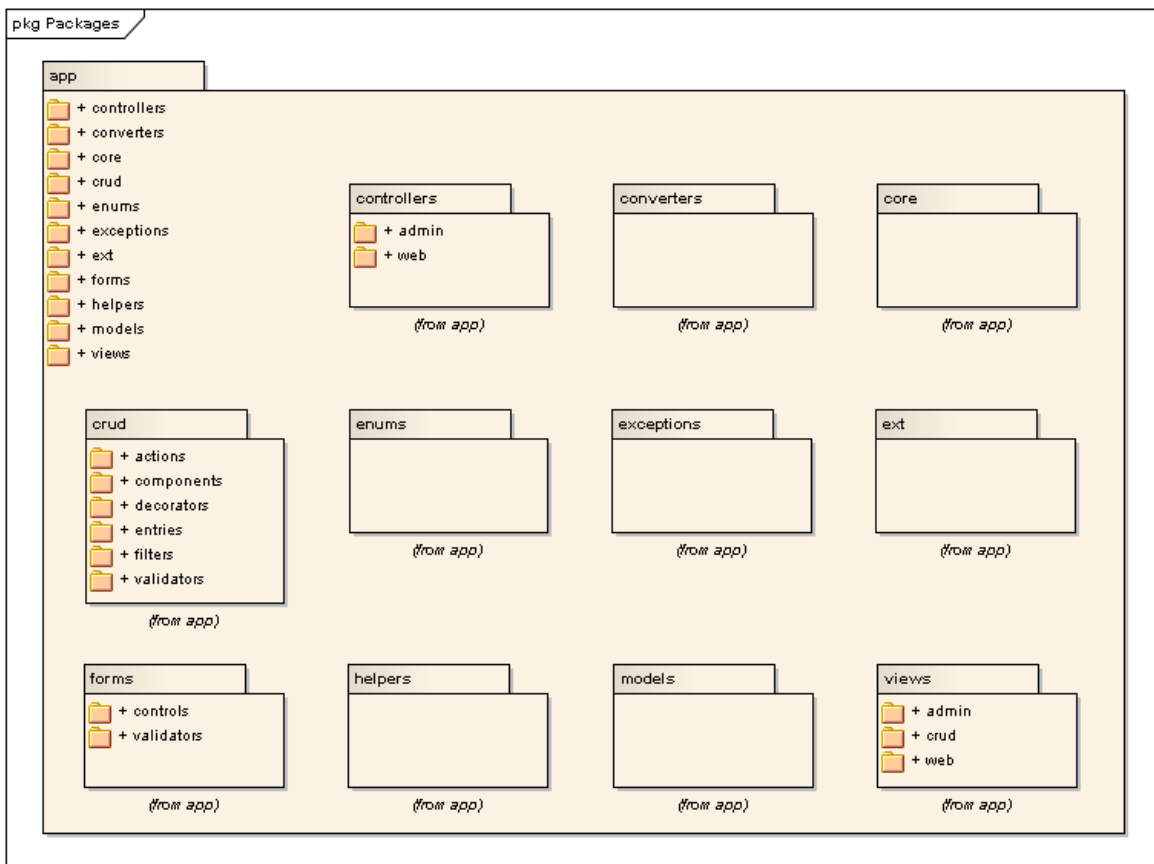


Figure 2.4: Urchin CMS structure at the package level.

The structure of the application is shown in figure 2.4. The most important package is core that contains fundamental classes of the WCFS framework. These classes will be

described in detail in section [Context and Core Objects](#). Other important packages are controllers, models and views that represent all parts of the MVC triad. The package controller contains front controllers and two sub-packages with common controllers, both for front-end and administration. The package models equals the database layer and is composed of classes working with database tables. The package with views contains layouts and templates for the whole application.

Other essential packages are crud, enums, and helpers, and converters. The package crud contains a complete library called crud for generation of basic user interfaces. The whole section [Crud Library](#) in the next chapter is dedicated to this invaluable library. Enums are similar to model classes, they encapsulate an application-wide sets of values, e.g. page states. Helpers provide various additional functionality, such as working with files, pagination, or sending e-mails. Converters serve a single purpose, to convert values to string representation and back again. The idea of converters has been adapted from JSF⁵ [22] technology.

The last three packages are exceptions, ext, and forms. The package exceptions contains all exceptions and the package ext external libraries. At this moment, there is only a single external library, PHPMailer [37]. The package forms contains additional classes for the dynamic form module. This module is much more complex than standard modules and therefore has assigned an independent package. The dynamic form module will also be discussed in section [Dynamic Forms](#) of chapter [Extending Modules](#).

2.5 Context and Core Objects

2.5.1 Context

Context is a static class that is accessible from any part of the application. This class is relatively small, its main responsibility is to provide access to all other core objects, all other core objects are accessed via this class. It also supports facilities for localization, logging and basic system messages. This concept was inspired by JSP technology and its interface ServletContext [23], although the realization in Urchin CMS slightly differs. Context was introduced in Urchin 2.0, three years after most core objects.

2.5.2 Session

Session object simply encapsulates session and provides its basic security. Session is a common feature that keeps application settings over otherwise stateless HTTP⁶ [18] protocol.

2.5.3 Request and Response

Request is object that provides access to headers and attributes of a HTTP request. In comparison with standard PHP mechanics, the request object also significantly simplifies working with default values or complex arrays in request. Request enables getting post, get and cookie values as well as request values and saving uploaded files. Response is a simple

⁵JavaServer Faces

⁶Hypertext Transfer Protocol

object that allows sending headers or redirections back to the client. Again, design of both classes was hugely affected by JSP technology.

2.5.4 Pool

Pool object serves a single purpose, it supplies data for view. Using pool is the only way how to pass data from the controller to the view. All variables (including nested arrays and objects) are automatically escaped for security reasons. Design of this object is influenced by the registry design pattern from [54].

2.5.5 Cache

Cache is a specialized object for caching database queries and other data. Internally, this object uses a hierarchical tree structure with indices for storing data. This tree structure is saved to the file system. Data are manipulated using tree nodes with labels, e.g. web → pages → cs. Tags for random access are also implemented but not used. Today, caching is used only for basic queries. Use of caching will be definitely improved in future versions of Urchin CMS.

2.5.6 Link

Link object is responsible for global handling of links (urls), both in the front-end and in the administration. These links could be static or dynamic and allow adding and removing parameters. Dynamic links automatically keep previous parameters unless explicitly removed, static links start without any parameters. In the front-end both rewritten and non-rewritten links are supported, current setting depends on system configuration.

2.6 Database Design

Well-designed and mature database architecture is the backbone of the Urchin CMS application. The whole database sub-system is fully transactional according to OLTP⁷ standards and normalized to both third and Boyce-Codd normal forms. The database layer is always bound to the chosen relational database management system (currently MySQL). On one hand, each database server requires its own database layer, on the other hand, the concrete database layer is tailored to the selected database for better performance. In contrast to the object-oriented application design, plain old arrays are used for data transfer. Unlike Java, this is a natural approach in the PHP programming language.

2.6.1 Database Layer

The database layer serves as the model part in the model-view-controller architecture and is based upon the table data gateway design pattern. Each class in the model is mapped to a single main table and possibly other related tables. This design allows effective separation

⁷Online Transaction Processing

of application logic and queries. Other advantage is relatively simple caching of database queries directly in model classes. A typical model class is stateless (has no properties) and provides all methods necessary to work with the underlying table. All models inherit `BaseModel` class that enables basic data-manipulation operations, e.g. `fetch`, `update`, `insert`, `delete` and transactional operations, e.g. `begin`, `rollback`, `commit`.

Queries used in the crud library are mostly generated while respecting foreign keys and indices. Both static and generated queries strictly use prepared (parametrized) statements for better performance and prevention of SQL injection and similar exploits. Basic caching of queries is already provided, advanced caching with dependencies is proposed for future implementation.

2.6.2 Database Structure

Database model of Urchin CMS core is composed of about 30 main and 15 additional tables (for lookups and translations). There are three major and several minor database parts. The major areas are the page axis, the component axis and the permission sub-system. Minor areas include search, locale, or logging. Figure 2.5 displays a conceptual database model with basic entities and relations of both axes. A prefix `a_` is used for core tables (`m_*` for module tables and `v_*` for views). The left side of the diagram shows fundamental tables of the page axis. These tables are `a_presentation`, `a_page`, `a_template` and `a_position`. The right side of the picture shows the component axis with tables `a_module`, `a_component` and `a_element`. Details about all three major parts will be discussed in the following chapter [Core Features and Modules](#) together with the application design. A complete conceptual model with all core entities is available in the appendix [Database Model](#).

2.6.3 Alternative Approaches

There are generally three common approaches how to work with a database in a web application. First one, a separated database layer, was already described. The second approach is to simply embed database queries into the programming code when necessary. This is common, but definitely not recommended approach as it mixes database queries and application logic. The last usual option is to use an object-relational mapping (ORM). This is especially common in the Java EE world, where products such as Hibernate [20] are used. More alternative ways for storing data are NoSQL database management systems, such as file-based databases, document-oriented databases, XML-oriented databases, or key-value storages.

2.7 Crud Library

Crud is an important built-in library in Urchin CMS. The library serves as an efficient tool for generating user interfaces and is used for managing most database data in the administration. Controllers inherited from crud library classes are in fact declared, no advanced programming is necessary except implementation of custom features. As the library name suggests, crud basically handles four typical operations with records: create, read, update, and delete. Beyond these elementary functions the crud library allows nesting

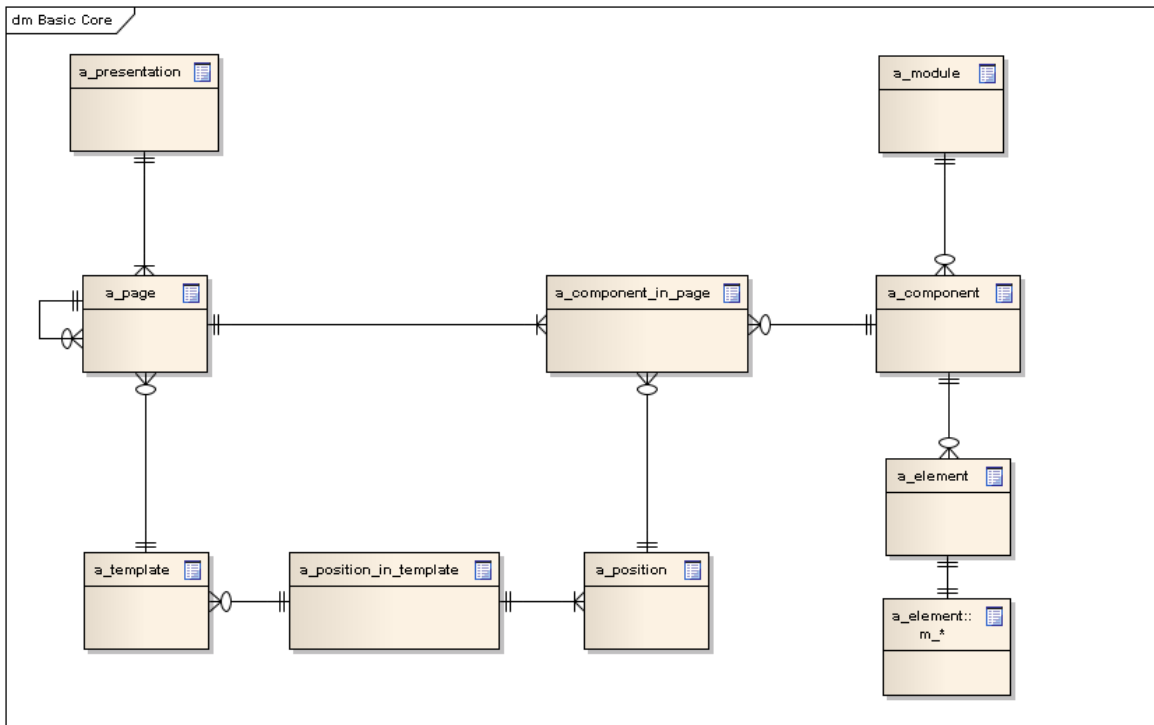


Figure 2.5: Database model with core tables and their relations.

of crud instances, filtering of records, validation of input, or adding custom functions, e.g. sending a mail. All operations very closely cooperate with Urchin permission system. In example, controller for managing presentations allows only view, detail and edit actions. Members of all users groups can preview presentations but only administrators are allowed to edit these records.

Crud works effectively with any database schema that has set up a single integer primary key for each standard table and properly uses foreign keys. Database relations up to M:N and M:N:X are supported. Queries in crud-based controllers are generated according to the definition in the crud controller class. Main advantages of the crud approach are rapid development, simple maintenance, reduction of duplicated code, and enough flexibility for adding custom elements or individual functions. There are also no generated files or forms, just simple and straightforward classes with definitions.

The crud library is composed of several main parts and sub-packages with various elements. See figure 2.6 for detailed structure of this important Urchin package. The main parts are common crud, element crud, cross crud, and matrix crud. Crud elements are entries, filters, components, decorators, and validators. The central part is common crud that handles single main table and several linked tables. Only a common crud instance might contain nested instances of any other type.

Element crud works exactly as the common crud, moreover it is adapted to manage elements. Elements are special database records used in content modules. Cross crud is used to manage M:N database relations, matrix crud to handle M:N:X relations. Nesting

of instances precisely follows the composite design pattern. Here, a common crud instance serves as the composite component while cross crud instance, matrix crud instance, or custom implementation create the leaf part. All mentioned parts will be discussed in the following sections.

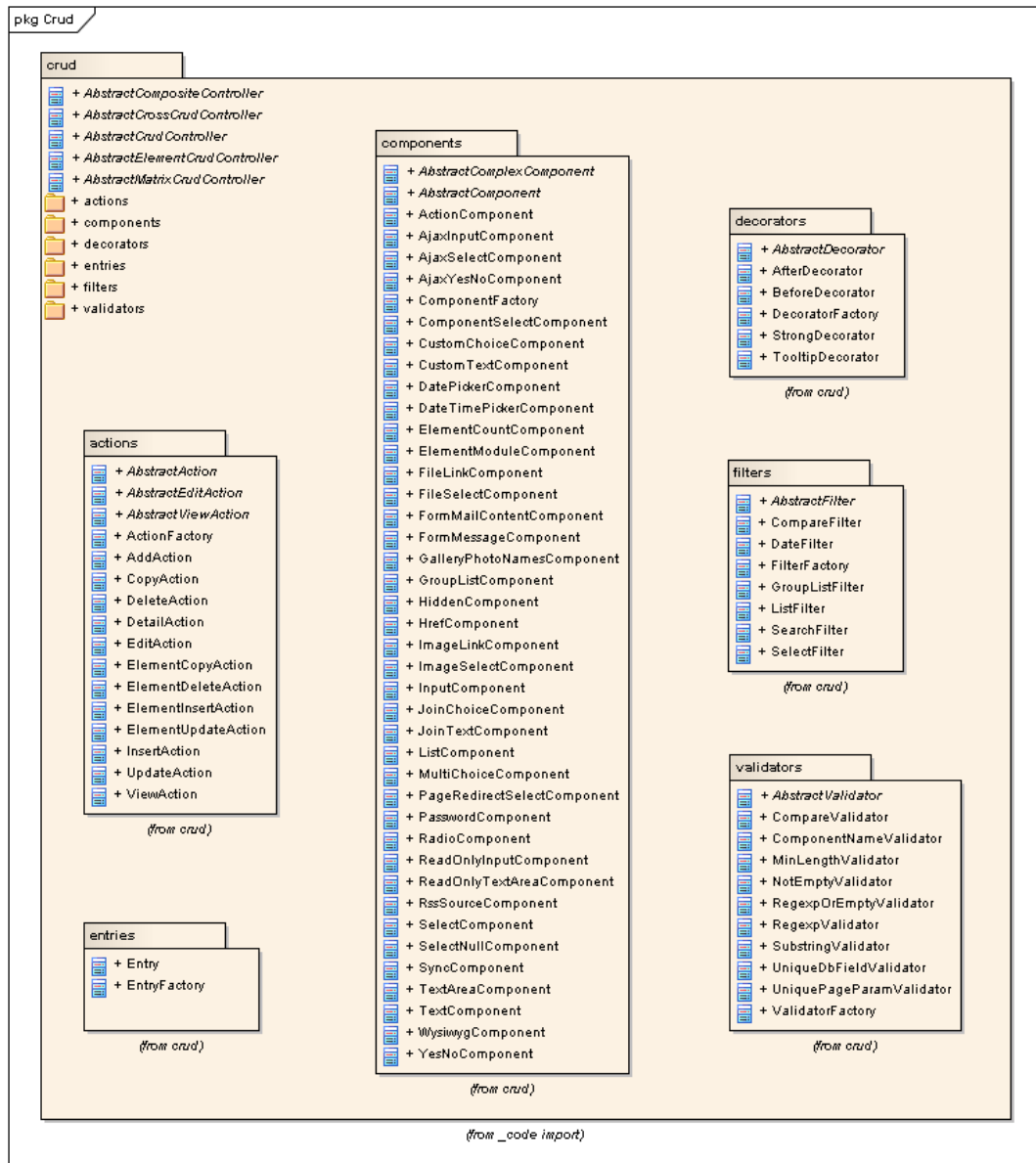


Figure 2.6: Packages and classes of the crud library.

2.7.1 Common and Element Crud

Common crud is the most important and most popular part of the crud library. A controller built upon a common crud instance manages records in a single main database table (or

The builder design pattern is applied for literally building common crud instances from entries, components, and other elements.

2.7.1.1 Entries

An entry is a basic unit used in the crud instance. Each entry serves as a container for a single component and validators. If assigned, a component tells entry how to display value. A database entry works with a single database column while simple entry does not require a database field at all. Entries are also responsible for pre-loading component data if necessary and marking components as required based upon assigned validators.

2.7.1.2 Components

Components present database values to the user in an appropriate format and also assemble these values back before updating or inserting a record. There exist about 40 various component types in the crud library. Some components only display values in the desired format, e.g. simple text, date/time, url link, or a value from a joined table. Other components are used in forms for editing records, such as input field, text area, radio buttons, or selection box. Specific components enable selection of files and images, working with elements, AJAX-based editing, or synchronization between fields. The action component enables calling custom actions implemented in the controller. This option is used e.g. for generating new password and sending it to the user in the administration. The hidden component allows adding default values for the database fields.

2.7.1.3 Decorators

Decorators are simple objects for enhancing displayed data independently of the assigned component. Several decorators exist in the crud library: to display text before or after component, to show a custom tool-tip, and to show bold text. Decorators strictly follow the decorator design pattern.

2.7.1.4 Validators

Validators are used to check user input in the edit and add actions. Each entry might include one or multiple elements of this type. Validators check user-provided data according to rules defined in the crud instance. The crud library has included many useful validators, for e.g. checking non-empty field, comparison with a custom value, regular expression based checks, or securing value uniqueness against database.

2.7.1.5 Converters

Converters modify values both before injecting them from the database to components and before putting values from components inside a form back into the database. There is huge difference between converters and components. Components only format data for rendering, they in fact do not modify anything. Converter package is located outside the crud library because converters are widely used in the whole application. Converters are used mostly to

encode and decode strings, transform date formats, or specifically treat database null values if required.

2.7.1.6 Filters

Filters are simple objects that serve a single purpose, filtering records in the view action. The crud library provides filters for searching by text, comparison with value or date, custom list or by selection from other database table.

2.7.2 Cross and Matrix Crud

The cross crud is the third important part of the crud library. It is used to manage M:N database relations (also called "cross" tables, hereby the name). M:N tables contain nothing more than two foreign keys to other database tables. A cross crud instance cannot exist on its own, it must be nested inside a common crud instance. The reason is that the value of the first column is taken from the parent instance and the user chooses values from the second column. The cross crud is much more simple than previous parts, it utilizes only two actions: the view action for displaying data and the update action for saving changes.

The matrix crud works similarly to the cross crud. It manages three-dimensional M:N:X relations. There is only one table of this type in Urchin CMS. This table is composed of exactly three foreign keys and is used in the permission sub-system. Again, the value of the selected column is set up while the user is allowed to work with two remaining columns. The user interface of the matrix crud therefore looks like a two-dimensional grid. The matrix crud also uses two actions, the view action and the update action.

2.7.3 Flexibility and Extensibility

The crud library is highly flexible and extensible. There are multiple ways how to extend the library. The easiest way is to create custom components and other elements. Implementation of components is very simple, these classes are usually simple and straightforward. The situation is similar with other elements, validators being the most simple. Many new elements have emerged this way during the development of Urchin CMS. The second possibility to extend the crud library is to use built-in callbacks. These methods are called before and after either delete, or insert, or update if used. All callbacks are provided with primary key value of the record and a corresponding model with data. Modifying the model is also possible. This option is used very often, e.g. for assigning default page group to the newly added page or working with positions in records works exactly this way.

Using the action component and implementation of this action in an inherited controller is the third way to extend the system. In example, sending a new generated password from the administration to the existing user uses this feature. Advanced programmers might also override existing methods with custom functionality. The element crud was developed this way, it is a significant extension of the original common crud with rewritten and enhanced modifying actions. In fact, an element crud instance works with a view instead of a table. This view is defined on exactly two database tables, a table with elements and a module-specific table that differs according to managed module.

Chapter 3

Core Features and Modules

3.1 Chapter Overview

The third chapter named [Core Features and Modules](#) discusses core features of a content management system. Core features are basic ideas that are parts of the global architecture and affect the whole application. First part of this chapter describes two fundamental parts of the system, the page axis and the component axis. The page axis deals with vertical partitioning of a website into presentations and a tree-based hierarchy of pages. The idea of component axis is based on modules, components and elements. The second part of this chapter discusses security and the permission system. The section about security identifies typical security problems of a web application and explains their prevention. The permission system is the third fundamental part of the Urchin application that is tightly connected to both the page axis and the component axis. The last minor part discusses several ideas of advanced content management.

3.2 The Page Axis

The page axis is the first of the core features used in the Urchin application. This section discusses vertical partitioning of a website into presentations and pages. The database schema in figure [3.1](#) displays relations between all tables that make up the page axis. This diagram shows basic tables for presentations, pages, and templates together with additional tables for languages, page groups, or parameters. All important parts and features concerning the page axis will be described in the following text.

3.2.1 Website

In the context of a web content management system, a website is a top-level structure that contains hierarchy of web pages with static or dynamic content. A website deployed with Urchin CMS works exactly this way. Concrete website runs on a single domain and uses a single installation of the system for its administration. Typical website contains one or multiple presentations, each with a customized page hierarchy. A presentation usually represents a language mutation or an independent part of the website, e.g. a product micro-site.

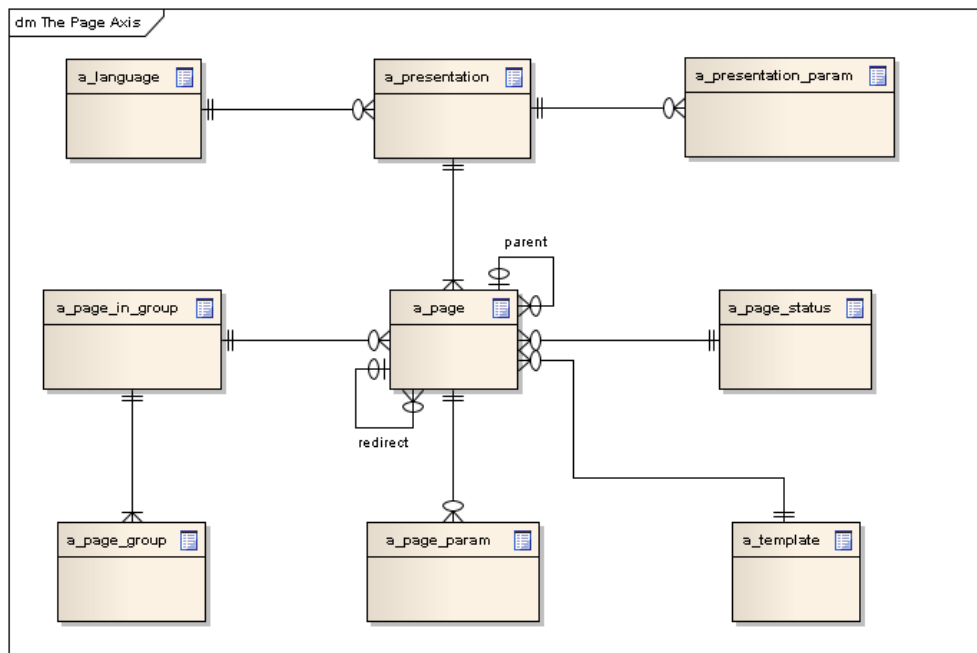


Figure 3.1: Schema of the page axis with presentation, pages and templates.

3.2.2 Presentations

A presentation is the basic part of the website, the website must contain at least one presentation. Each presentation has assigned a language and is composed of a hierarchically arranged tree of web pages. Presentations might be either used as language mutations of the website, e.g. English and Czech version, or as independent sites of the website, e.g. company and product presentation. Combination of both approaches is also possible. Different presentations in the front-end are recognized using a unique url segment. In the future, running each presentation on a different sub-domain will be supported.

In the administration, basic SEO¹ settings for presentations are provided. The website administrator is allowed to set up presentation title, pretty url, keywords and descriptions. All of these settings are shared across pages in the same presentation. Presentation parameters enable adding custom variables in the presentation scope. This feature is very useful for custom modules or functionality, but is not used in default application. In the administration, the user always selects a single presentation to manage. This severely reduces complexity of the user interface.

3.2.3 Pages

Pages represent the basic structure of a website. Each presentation contains its own different set of pages forming a tree-based hierarchy. A typical page has zero or more sub-pages according to its position in the hierarchy. Main purpose of pages is to present website content

¹Search Engine Optimization

and allow user interaction with the website. The page content is managed using components and elements. These units will be discussed in section [The Component Axis](#). Each page uses a single template with multiple pre-defined positions that serve as containers for components. In the administration, pages for currently selected presentation are managed in a tree view. This tree view allows user to perform all basic actions for modifying both pages and content elements. Page groups are used in the permission management, each page by default belongs to the main page group.

The page administration provides many additional and useful features. Basic settings include page title, heading, pretty url and flags for displaying page in the main menu and in the site-map. It is also possible to redirect page to another page in the same presentation (cyclic redirections are not allowed). This option is commonly used to redirect to the first sub-page in the lower level of hierarchy. Each page might have planned publication for a chosen interval or belong to the restricted zone. Pages in the restricted zone are available only to the logged users in the front-end. Similarly to presentations, pages allow user to edit basic SEO parameters, e.g. pretty urls, keywords and description. Changing position of a page is possible manually or using a drag-and-drop feature in the tree view.

The diagram in figure 3.2 shows a simplified structure of a company website. This website is designed using the described principles. The website has two different presentations that serve as English and Czech language mutations. Each presentation has several pages, the structure of pages differs with the presentation. In the diagram, the English presentation has five different pages: home page, news, services, references, and contact while the Czech mutation has only three pages: home page, services, and contact (all system pages are omitted).

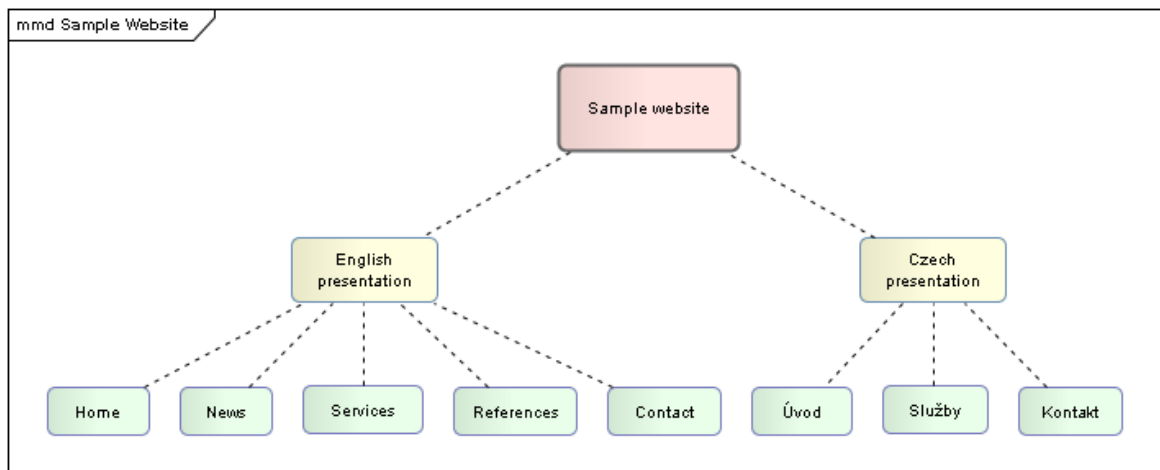


Figure 3.2: A sample website with two presentations and several top-level pages.

3.2.4 Templates and Positions

Templates are important for displaying page content. A template represents page layout and is very closely related to the graphical design of a website. In example, a website has

three graphical layouts: a home page, a common page, and a product page. Each layout corresponds to a different template. The template usually contains one main and several other positions for assigning components. The number and location of positions depends on the concrete template and is fixed. Templates and their positions are designed before the website content is created. These settings are static and cannot be changed by the user via administration (manual change in the database is possible without any harm). Each page uses a single template that is selected during the page creation. Available positions, as well as assigned components and their elements are visible in the page detail in the tree view. All templates internally use a built-in PHP templating system [36], which is the most minimalistic solution.

3.2.5 Page Parameters and System Pages

Page parameters enable setting up custom variables within the page scope. With appropriate permission, the user can edit these parameters in the administration. There are two different types of page parameters, custom and system. Both types work similarly, they are key-value pairs assigned to the selected page. Custom parameters are used for implementing custom modules and functions. In every presentation, there are five system pages, distinguished by system parameters. System pages and parameters cannot be deleted because they provide fundamental functionality. The default system parameters are:

- `is_homepage = 1`
- `is_sitemap = 1`
- `is_fulltext = 1`
- `is_error_404 = 1`
- `is_error_403 = 1`

The home-page serves as the default page in the presentation, the site-map displays a simple tree with pages and links. The full-text page shows results of built-in search. The error pages are shown to the visitor if he requests non-existing page (404 Not Found) or is not authorized to view the requested page (403 Forbidden).

3.3 Internationalization and Localization

Urchin CMS is a completely multi-lingual application. Many languages are supported both in the administration and in the front-end. Each language has assigned a locale for formatting strings and adjusting user interface. All files and translations in the system use the UTF-8 [46] encoding. The Czech and English languages are default for the administration, the Czech, English, German, and Swedish languages are available in the front-end. Adding a new language to the front-end requires translation of all terms and putting them to the configuration file. This new language must be added to the `a_language` table and selected in the presentation settings. Internationalization and localization applies to all strings, such as titles, labels, and feedback messages.

3.3.1 File-Based Translations

From the technical point of view, two different approaches are used for translating terms. The first technique is very simple and flexible. It uses a translation method of the Context class. This method takes a string as an input and returns a translated term. The input string also serves as a default value, no artificial keys are necessary (but sometimes used for feedback messages). Hence this method is suitable only for short terms up to a single sentence. Values are looked up in the array with translated strings that serve as a simple key-value storage. Dynamic parameters in the translated terms are also supported. This system is hugely influenced by the QT framework [41] for C++ and its internationalization system.

3.3.2 Database-Based Translations

The second method is used exclusively for database-stored system terms in the database. For each database table and column using this approach, an additional table exist. This translation table appends suffix `*_ext` and contains translated strings along with a foreign key to the language table. In example, module or state names are translated using this method. Terms translated this way are never modified by the user and are often cached to counter an additional join required to retrieve these records. Moreover, this method is used only in the administration. Other popular approaches for translating strings are `gettext` [12] or storing all terms in the database thus allowing the user manipulate this data.

3.4 The Component Axis

The component axis represents the second most important core feature employed in Urchin CMS. This concept deals with horizontal partitioning of a website that is implemented using a unique system of components. Components are derived from modules and serve as simple containers that are assigned to web pages. A typical component contains one or multiple elements. An element is basically a specific database record with specific parameters and behaviour, such as a simple text, an article, or a web form. Elements represent the lowest level of website content. Diagram in figure 3.3 shows all database tables of the component axis and relations between these tables. Main tables work with modules, components and elements. Remaining tables enable linkable modules and store additional information, such as module templates or parameters. All major parts of the component axis will be discussed in detail in the following text.

3.4.1 Modules

A software module in Urchin CMS is an independent unit of code that serves a single purpose. Each module always contains one or more administration controllers and a database script. Most modules employ additional files, such as model classes for all database-oriented modules or templates and web controllers for front-end modules. Regardless of type, each module has a public interface and hidden implementation. Public interface is used for communication with other parts of the application whereas all internal operations are separated

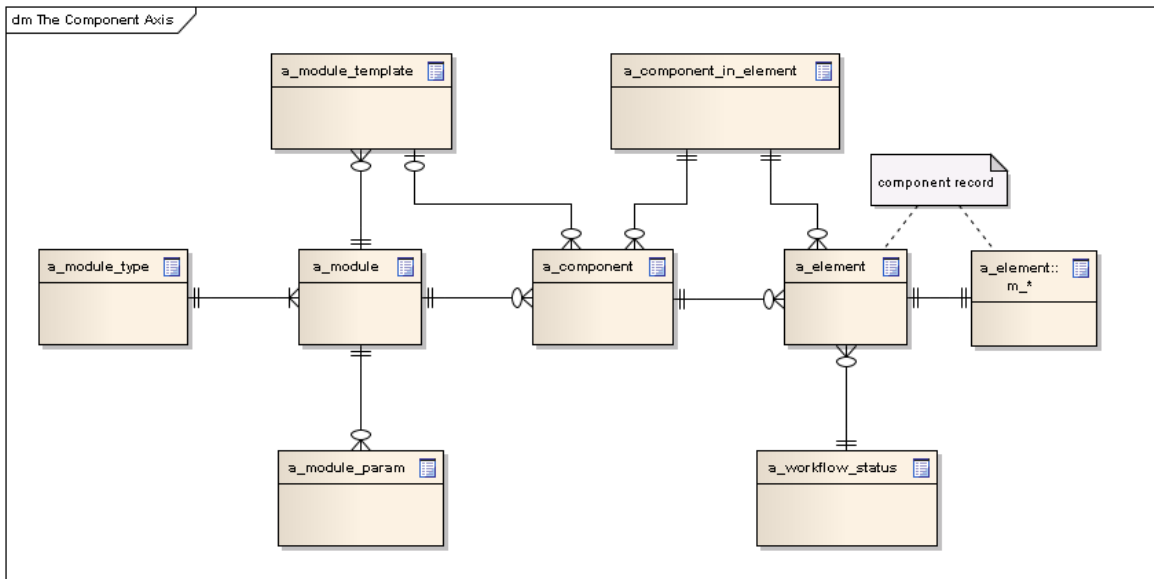


Figure 3.3: Schema of the component axis with modules, components and elements.

from other system units. This design serves both flexibility and maintainability of the Urchin application. Installing a new module is simple and has two phases, copying files and running the database script. The installation script connects the new module to the system, adds new database tables if required and sets up desired permission for default user groups.

Two basic types of modules exist in the system, called system and web modules. System modules form integral part of the system that is used exclusively in the administration for internal tasks and settings. System modules never contain sub-modules and therefore use only one controller per module in the administration. Content modules are used in the front-end for presenting website content to the visitor. In contrast to the system modules, more complex content modules consist of multiple sub-modules. Content modules are also managed from the administration utilizing crud-based or custom controllers.

3.4.2 System Modules

System modules are used solely in the administration and allow managing system settings, permission and various other features. Most system settings are restricted only to the administrator group, not ordinary editors or supervisors. Only user settings are available for all user groups. Main areas handled by the system modules are shown in the following list.

- permission - users, groups and access rights
- modules - internal settings and parameters for modules
- caching - manual purge of cached data
- lookup tables - system states and enumerated values

- user log - logging of user actions in the administration
- user preferences - personal settings, favourite links, internal contact form

3.4.3 Content Modules

Main purpose of content modules is to enable administration of the website front-end. Content modules are used to manage website structure and content from the administration. As denoted in the previous text, some content modules contain multiple sub-modules. Content modules are divided into two categories according to the module type. These two categories are application and element modules. Complex modules commonly contain sub-modules of both types.

3.4.3.1 Application Modules

Application modules work in the same manner as the system modules. The administration of system modules is exactly the same as for system modules and uses similar common crud-based or custom-implemented controllers. Unlike element modules, these ones do work only with standard database records, not elements. The application modules either provide basic data for the front-end or support element modules in delivering content. The following list shows most relevant areas that use application modules outside the content module scope. Application modules cooperating with element modules will be discussed in the following text.

- the page axis - presentations, pages, templates, positions
- the component axis - components and elements
- localization - languages, locale, translations
- files - file and image manager
- indexing - transformation of content for full-text search
- clients - front-end users and groups

3.4.3.2 Element Modules

In contrast to application modules, element modules are used especially to provide content of various type to the front-end. These modules work with special database records entitled elements. Elements share common features but differ in module-specific options. Element records are always managed using a controller that is instanced from the element crud. Administration of elements closely cooperates with permission sub-system and content workflow. Technical details about elements will be described in a separate section.

A simple content module always equals a single element module. A complex content module is usually composed of a single element sub-module and one or more application sub-modules. In this case, the element module is responsible for publishing content on the

website while the supporting application modules take care of additional information. See figure 3.4 that shows examples with structure of several content modules. In the diagram, three of basic content modules are present. These are Article, Event, and QuickContact module. Component entities in figure represent whole modules, class entities mirror both administration controllers (sub-modules) and database tables.

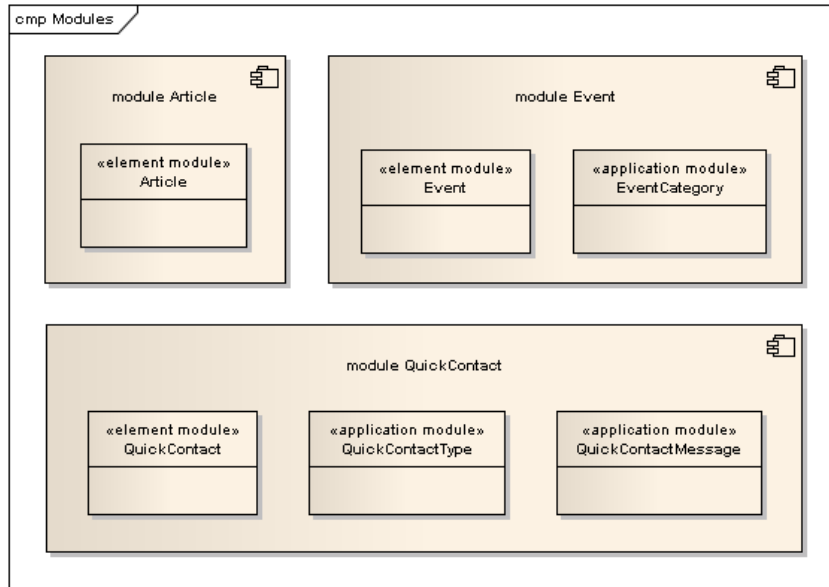


Figure 3.4: Organization of content modules into element and application sub-modules.

3.4.4 Components

Components are derived from the element modules and used for publishing content on the website. The module serves as abstraction while components realize this abstraction. In example, the module Articles has derived two components entitled Sport and Business with articles about these topics. A component serves as a simple container for elements and is used to manipulate these elements as a group. The maximal number of elements allowed in the component is defined in the element module. There are modules that allow zero, one, or unlimited number of elements per component. Modules with unlimited number of elements are used to group similar entities, e.g. news, articles, or events. Modules with a single element represent content without records, e.g. simple text, any forms, enquiries. Modules without elements do not have any content and are used only for default system pages with search results and a site-map.

Components are being assigned to website pages. As described before in the previous section [The Page Axis](#), every page has defined a template with positions. These positions work as place-holders or slots, one position for each component. In example, a page Contact contains components with a short text, a map, and a contact form on three different positions. All menus are built-in and do not require components. Components do not have any settings beside title, internal description and a voluntary module template. All settings are

managed on the element level, a component either is or is not assigned to the page. Components are very flexible, the same component might be assigned to multiple pages, even across the presentations. This possibility prevents unnecessary duplication of the website content.

3.4.5 Elements

As already denoted, elements are specially designed database records. Elements represent the low-level layer of website content. From the technical point of view, each element is composed of two parts, the element part and the module extension. Both parts are connected via a 1:1 database relation using the class table inheritance design pattern [9]. The element part is always stored in the `a_element` table and the module extension in a module-specific table, e.g. `m_news` for news elements. The diagram in figure 3.5 displays relations between the `a_component` table, the `a_element` table and three module-specific tables.

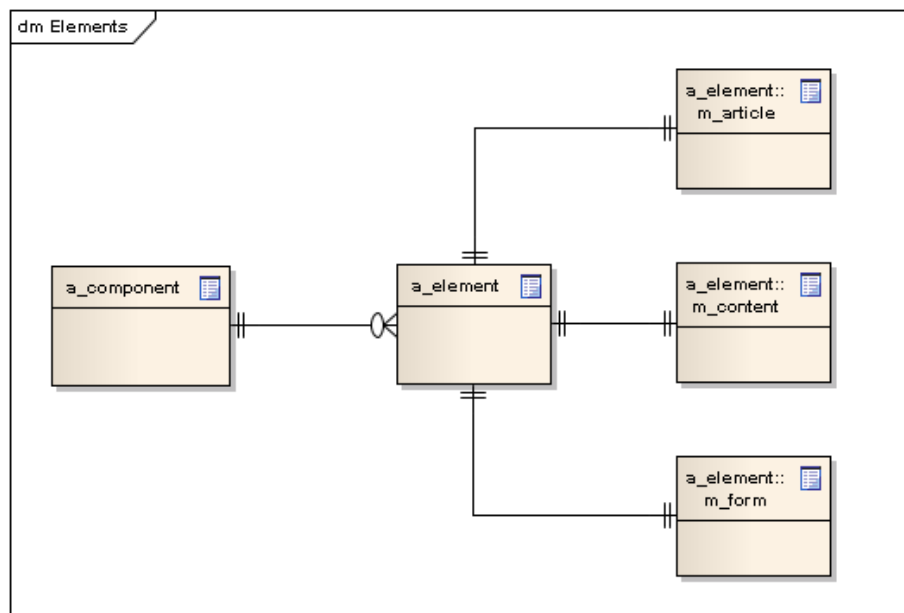


Figure 3.5: The relation between the element table and module-specific tables.

The administration of elements provides a handful of useful settings and features. Managing element title and planning publication is available to each element. This includes setting up dates and the work-flow facility. The dates define the interval when the element is visible on the web, the content work-flow will be discussed in section [Content Work-Flow](#). Other settings and options vary with the module. Many additional content-oriented features of the application are also handled on the element level. This includes content indexing for the full-text search facility, content approval, online preview or logging users' actions in the administration.

3.4.6 Module Templates and Parameters

Module templates is an optional front-end feature of Urchin CMS. In the former versions of the system, each module had exactly one set of templates for the front-end. All components of the module therefore had to look exactly the same on the web. In contrast with this situation, module templates allow the user to choose a set of templates separately for each component of the same module. All additional files for templates must be created and installed before the selection is available.

3.4.7 Linkable Modules

Linkable modules add an additional and slightly different concept for handling front-end content. Instead of assigning components to positions on the page, components of the linkable modules are connected to elements. A linkable component cannot be published on its own, the parent element is mandatory. Concept of linkable modules enables additional functionality for elements, such as adding a photo gallery, a discussion or voting, or adding a map with coordinates. In addition to the graphical appearance, the assigned component has information about the parent element. At the moment, this feature is experimental with only the Gallery module available. For logical reasons, each element might have assigned only one component per linkable module, e.g. one discussion and one gallery but not two discussions.

3.5 Security of a Web Application

This section lists typical and wide-spread security problems of web applications. The following text discusses details of four common security problems according to the OWASP² project. The investigated areas are: SQL injection, cross-site scripting, authentication and session management, and cross-site request forgery. Each problem is shortly described along with its common causes and solutions, both general and Urchin-specific. Other types of security problems are omitted in this work. The most recent stable list of top 10 security risks is available at [29] and the most present but pending list accessible at [32] for details.

Additional security features in the Urchin application are related to either the permission sub-system or specific modules. Details about permission and access rights are discussed in the following section entitled [The Permission System](#). The most remarkable content modules are described in chapter [Extending Modules](#) including security aspects of these modules.

3.5.1 SQL Injection

SQL injection [31] is a most common but under-estimated type of attack in the web environment. It is not difficult to perform this kind of exploit even for an average programmer. A successful attack often results in data loss, theft, or corruption. A software application is vulnerable to the SQL injection if its database queries allow putting untrusted data in the query construction. Untrusted data come usually from the user's input or a similar

²The Open Web Application Security Project

source. The injection occurs when the data are incorrectly escaped or directly concatenated into the query. Other types of injection also exist, this includes exploiting operating system commands or various application parameters.

The solution for this problem is to never trust the user's input and use a secure API³ to prevent the vulnerability. The user's input must be always validated and sanitized before using values in the application. The validation checks e.g. the data type of the value, allowed interval or white listing. A good API must enable separation of database queries and their arguments. Well-known libraries for the PHP language that satisfy these requirements are PDO⁴ [39], ADOdb [1], and MySQLi [38].

In the case of Urchin CMS, validators are widely used for controlling input, input values are sanitized and the PDO library is employed for building database queries. The whole model layer uses this library for building prepared queries. A tricky part comes with the crud library that in addition uses dynamic queries. Crud queries include parameters such as table or column names that cannot be parametrized this way. But this is not a security risk because table and column names are always statically defined in the crud instances so they never come from the user's or other dynamic input. The limit clause cannot be parametrized either, a simple solution here is to check whether the value is a valid integer.

3.5.2 Cross-Site Scripting

A cross-site scripting (or XSS) vulnerability [30] occurs when a non-escaped value is output to the user. This value comes from the user's input or another source and contains a text-based script that exploits the website. The attack then performs a malicious action, e.g. redirects the user, steals his cookie or inserts unwanted code into the page content. In example, a stolen session identifier from the cookie is often used to further exploit the site using a CSRF⁵ attack. The solution for this security problem is to correctly escape all untrusted values before propagating them into the browser. The concrete escaping depends on the context, e.g. a safe string for HTML output differs to a safe string in the JavaScript context.

Urchin CMS employs this approach on the view level. Templates include slots that are used to output variables in the HTML context. All template values are sanitized in the Pool object before sending them to the template. Inserting cascading style rules or client scripts is not allowed in any content field and automatically excluded from rendering. Displaying HTML is only possible in fields that use the TinyMCE [44] WYSIWYG editor. The Link object escapes all parameters in automatically generated links.

The caveat of the WYSIWYG content editing is that it naturally requires modifying and rendering of the HTML code. The automatic escaping procedure therefore cannot be utilized. WYSIWYG content editing is allowed only in the back-end by editors and all actions in the administration are logged. This protection is far from being perfect, but sufficient. More sophisticated solutions also exist, in example the HTML Purifier project [17], lightweight mark-up languages, such as BBCode [4], or a mark-down syntax. The first library is a verified solution but is a total overkill comparing its size to the Urchin application.

³Application Programming Interface

⁴PHP Data Object

⁵Cross-Site Request Forgery

The second one is a possible option, but offers only a reduced comfort and set of functions for the user.

3.5.3 Authentication and Session Management

This section shortly describes security vulnerabilities tied to authentication, session, and account management. The attack of this type is possible in several different ways. Plain-text passwords in the database present severe risk if the attacker gains access to the database. The authentication process might be exploited if non-secure or weak hash functions are employed. The session is vulnerable to attacks if the session identifier is exposed in the URL, an application is vulnerable to XSS attacks, or the session is not regenerated after significant actions, e.g. logging in. A stolen session ID enables exploiting user accounts and data, especially in the case of privileged accounts.

The Urchin application uses several steps to prevent this kind of attacks. This prevention begins with the authentication process. Passwords in the database are stored in an encrypted form and salted to prevent a rainbow table attack. In the current minor version, the former hash function SHA-1 [11] used for storing passwords has been replaced with the stronger SHA-2 variant. Functions for account management do not expose vulnerable information, such as user identifier, login, or mere existence of the user in the database. All actions in the administration are logged, including authentication.

The session identifier is changed after significant actions, such as when user logs in. The log out link is provided to explicitly log out the user and remove his session data. If the user closes browser instead of logging out, the session time-out applies after the defined interval. As described in the previous text, additional XSS protections are utilized to prevent stealing the session identifier. In addition, a completely custom session handler and an optional use of SSL⁶ connection for the login form are planned to further improve security.

3.5.4 Cross-Site Request Forgery

The cross-site request forgery (or CSRF) [33] is the last type of attack listed in this text. This attack occurs when the hacker forges a HTTP request. The user is then tricked to submit this request, generally using social engineering methods. In example, he unwillingly submits this request via the image src attribute that includes a malicious link. XSS exploits are also used for triggering this attack. The success of the attack depends on predictability of the forged request. The simple but sufficient solution for this problem is to generate a unique and non-predictable token for each request.

Urchin CMS employs exactly the previously stated method. Requests that modify data in the application are always sent via the POST method. This includes all forms in the application. The token is generated before the page is loaded, so it is unique for each non-AJAX request and shared for multiple AJAX requests on the same page. On the other side, simple links do not need this security mechanism because they do not change any data in the application.

⁶Secure Sockets Layer

3.6 The Permission System

This section of chapter [Core Features and Modules](#) discusses the permission system that is the last of the three fundamental concepts of the Urchin application. The first part of the text explains main ideas behind managing user accounts and permission in a web application, e.g. access control, authentication, authorization and access rights. The second part describes application of these ideas in Urchin CMS. This part describes user groups, managing access in the system and differences between the administration and the front-end. The last short part discusses planned permission management on the front-end.

3.6.1 Authentication and Authorization

Authentication and authorization are the two most important mechanisms concerning controlled user access in an online application. Authentication is the process of identifying the user and verifying his identity. Authentication systems depend on a unique information known only to the system and the user. Such information is usually a common password. Other methods are also possible, e.g. using a client certificate or a fingerprint. The user is challenged to provide his unique information in the login phase.

In contrast to the authentication, the authorization mechanism is used to determine the level of access the user should have assigned. Authorization mechanism is utilized after the user is successfully authenticated. Based on the permission settings, the application authorizes the logged user to perform particular operations and access system resources. Both mechanism could be either separated, e.g. using different software or servers, or tightly coupled, e.g. inside the same application. The latter option is common in web applications. Figure [3.6](#) shows a simplified login process for a web application with both authentication and authorization facilities.

3.6.2 General Access Control

Every content management system requires some kind of access control to recognize the user and grant him permission to work with the application. The most primitive systems use a simple authentication mechanism that checks user credentials against static configuration. User accounts in such an application cannot be added or edited. More complex systems work with users. Access rights are connected to the users and this kind of application supports multiple user accounts.

Advanced applications employ a more sophisticated system of permission for the administration, including user accounts and groups. User groups serve as roles, e.g. admin or editor. Access rights are therefore granted to the groups instead of users. User groups are effectively used both to combine access rights from more than one user group and manage permission for many users at once. User groups could be either fixed or dynamic, or both. In addition to the administration, many web applications also manage user access to the front-end.

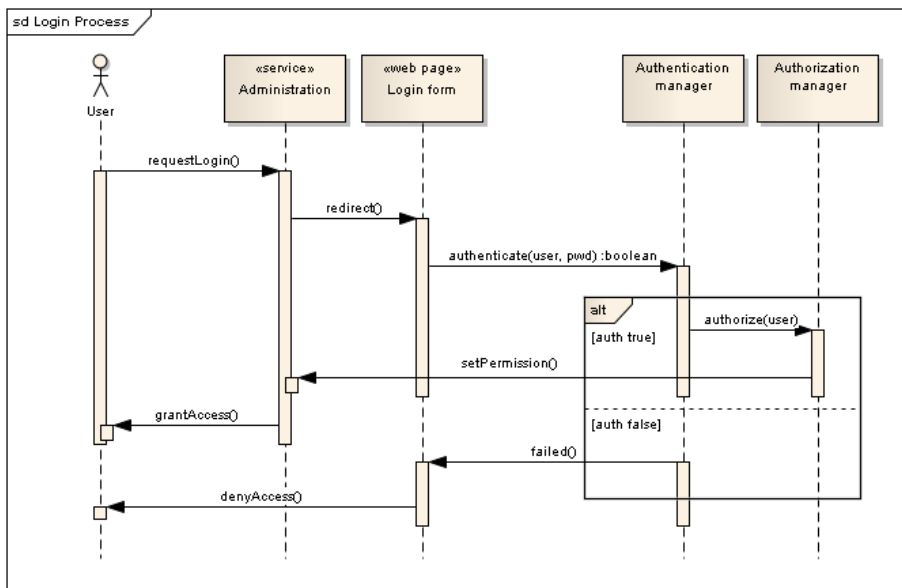


Figure 3.6: Login process in a regular web application with authentication and authorization.

3.6.3 Action and Data Permission

There are two most important models to manage permission for web applications, either an action-based approach (also called action control list) or a data-based approach. Non-trivial web applications and content management systems combine both methods up to some extent. The action-based approach works with roles, objects and operations while the data-based approach works with roles and records. User roles are represented by user groups and accounts as already denoted in the previous text.

The action-based model defines objects and operations. Objects represent various entities that have attached a set of operations. Entities are in example files, modules, or resources. Each entity-operation pair has defined the basic access, either allow or deny. User roles have assigned these access rights as required. The data-based approach allows fine-grained permission management within a single entity, such as a database table. This method enables differentiated level of access to records of this table that would not be possible with the action control list.

3.6.4 User and Groups

The administration of the Urchin application uses a combination of security mechanisms described in the previous text. Users belong to zero or more user groups and access rights are associated with the user groups. Four user groups with predefined access rights are provided with the installation, as enlisted in table 3.1 along with summary of permission. Access rights for both default and custom user groups are completely customizable. New groups and accounts might be added later.

user group	visible	used by	summary of access rights
editor	yes	client	create and edit content
supervisor	yes	client	as editor + publish content, manipulate pages & files
admin	yes	client	as supervisor + manage accounts, system tasks
root	no	developer	as admin + manage user groups and permission

Table 3.1: Default user groups available in the Urchin application.

The permission system includes three independent levels of controlling user access, these are modules & actions, presentation and pages. Figure 3.7 displays a database model with all important areas of the permission system. Details of all levels of permission will be discussed in the following text. Access rights to all areas are denied unless explicitly assigned to the group. Even the root user group does not have unlimited permission.

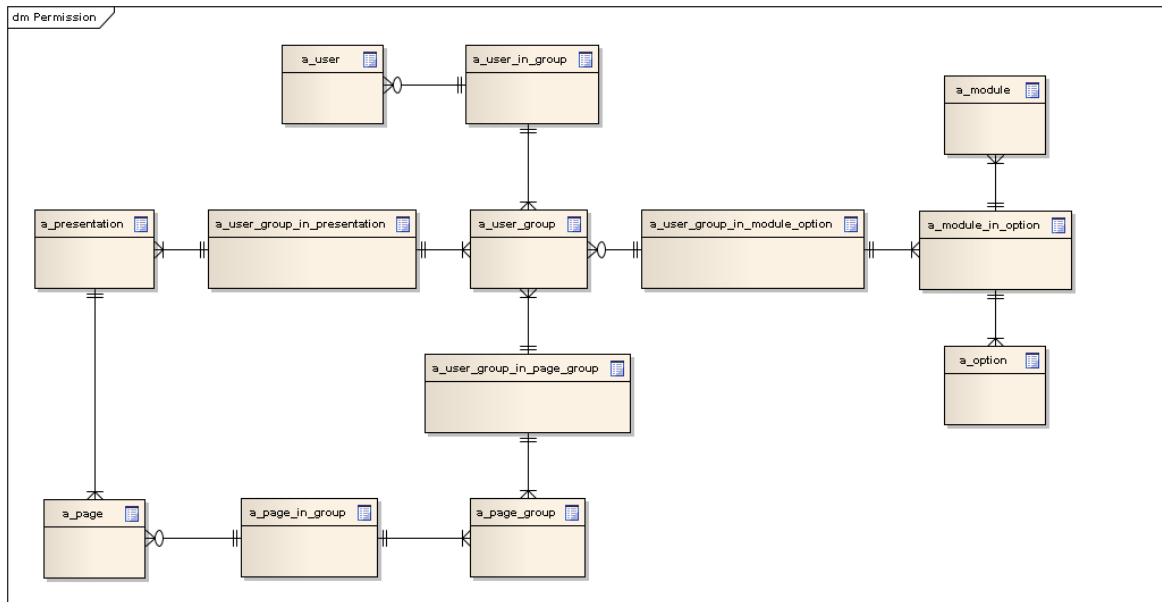


Figure 3.7: Database tables of the permission system and their relations.

3.6.5 Permission for Modules and Actions

Implementation of access rights directly follows the already described action-based approach. Urchin modules serve as objects and options as operations performed on these objects. There are six defined options in the Urchin system, as explained in table 3.2. Each module has assigned one or more options (the view option is always present). The module-option pairs are immutable and provide the basic matrix for assigning permission to the user groups. In figure 3.7, the `a_module_in_option` table contains all available pairs and the `a_user_group_in_module_option` table stores access rights assigned to the user groups.

access option	common use
view	view list of records
detail	view detail of record, access nested crud instances
edit	edit existing record
add	add a new record
delete	delete an existing record
approve	publish element (element modules only)

Table 3.2: Options available to the modules.

module	type	assigned options
Caching	system	view, edit
Content	element	view, detail, edit, add, delete, approve
Language	system	view, detail
News	element	view, detail, edit, add, delete, approve
Page	application	view, detail, edit, add, delete
Presentation	application	view, detail, edit
Tree (of pages)	system	view

Table 3.3: Sample modules of various types with assigned options.

Assignment of options depends on the module's purpose. Simple modules have only view while more complex modules employ up to all five basic options. The approve option is connected to the content work-flow and therefore available only to element modules. Content work-flow will be further described in the last section of this chapter. Table 3.3 shows sample modules together with available options and additional information.

Common crud works with all five basic options, element crud with all six options, and cross crud only with view and edit options. Main crud actions are directly mapped to the options, the crud instance is adjusted according to the assigned access rights. Several actions (methods) usually share a single access option, both in crud-based and custom controllers. Access rights are managed using the matrix crud, as seen in figure 3.8 that displays part of these settings. This is the only use of a matrix crud instance in the application.

3.6.6 Permission for Presentations

Data-based access rights in the Urchin application are managed on two levels, the presentation level and the page level. This feature is not available nor planned for e.g. components and elements although custom implementation is possible. Presentation-based access rights are very simple. If the user has sufficient permission, he can perform actions in the presentation, otherwise he cannot. As displayed in figure 3.7, the table `a_user_group_presentation` stores these settings. This settings is useful when different users are responsible for different presentations. Users are allowed to access all presentations according to default settings, so this permission is in fact suppressed if not required.

The screenshot shows the 'Administrator groups' management page. The 'Permission for actions' table is as follows:

Module	View	Detail	Edit	Add	Delete	Approve
Administrator group states	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				
Administrator groups	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>			
Administrator states	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				
Administrators	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Administrators in administrator groups	<input checked="" type="checkbox"/>		<input type="checkbox"/>			
Articles	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Components	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
Components in elements	<input checked="" type="checkbox"/>		<input type="checkbox"/>			
Components on pages	<input checked="" type="checkbox"/>		<input type="checkbox"/>			
Contact form	<input checked="" type="checkbox"/>					
Content	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Elements	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	

Figure 3.8: Managing access rights to actions in the detail of the user group.

3.6.7 Permission for Page Groups

Permission for page groups is a mechanism similar to the presentation access. The idea behind this feature is to separate responsibility for pages of a large website into smaller groups. Page groups are then assigned to the user groups. Each page group could contain pages from different presentations. The table `a_user_group_in_page_group` is used to store these settings, see figure 3.7 for details. As well as for presentations, use of this feature is optional. Each page belongs by default to the main page group, this cannot be changed. The main page group is pre-selected for all predefined user groups. Additional page groups are then added for advanced page management.

3.6.8 Additional Features

This short text describes two minor features connected to the permission system. The first must-have feature is detailed logging of all modifying actions in the administration. Modifying actions change data in the database, e.g. edit or delete. Logging in and out of the application is also tracked. Non-altering actions are not logged to prevent excessive size of the log. Each log record includes information about the user, module and action performed, unique record identifier, IP address and the time of change.

The second feature is the built-in soft delete. The soft delete sets record status to deleted (or similar) instead of physically deleting the record. This option avoids potential data loss and might be also useful in the future for content versioning or a recycle bin facility. There are also two minor drawbacks of this approach, the larger size of the database and the

user group	registered	access rights	note
visitor	no	view public pages	any non-registered visitor
registered	yes	browse client zone	
member	yes	browse client zone	for custom options

Table 3.4: Front-end user groups available in the Urchin application.

necessity to exclude those "deleted" records in queries, such as counting records. Complexity of queries is slightly reduced using views for all content modules.

3.6.9 Front-End Permission

Even though the front-end registration and user have not been yet implemented, basic ideas for front-end user management already exist. The same concept of users and user groups will be utilized for the front-end. Two user groups are planned, registered user and member. Table 3.7 shortly summarizes all front-end user groups. Page settings already enable putting the page into the client zone. Users on the web will register and log in to the system using a pair of new modules. Those that do not register will remain visitors. Visitors are allowed to browse any public page of the website.

3.7 Additional Features

This short section briefly describes three minor but important areas connected to the core features, especially to elements and permission management. These topics are content work-flow, content preview and personal settings. The content work-flow describes the built-in facility for approving content while the content preview enables previewing elements before they are published. Personal settings is about customizing user account and settings.

3.7.1 Content Work-Flow

A simple work-flow management is an integral part of the Urchin application since its first version. The work-flow is realized on the element level, so it is automatically applied to every element module and therefore to any publishable content of the website. Pages are not part of this sub-system because they in fact do not store any content, only contain positions for components with content.

The element life cycle includes five possible states, as displayed in figure 3.9 featuring a comprehensive state diagram. Element work-flow states are closely coupled with the permission system and its approve option. There are four states used in the work-flow: the edited, the waiting, the approved, and the rejected state. The user can switch between element states as depicted in the diagram if he has the corresponding permission. The delete state is not a part of the work-flow management, it is used only for the soft delete feature. Softly deleted elements are never displayed in the administration.

Newly created element have always assigned the edited state. If the user has assigned the approve option for a concrete module, he can change element of this module to any

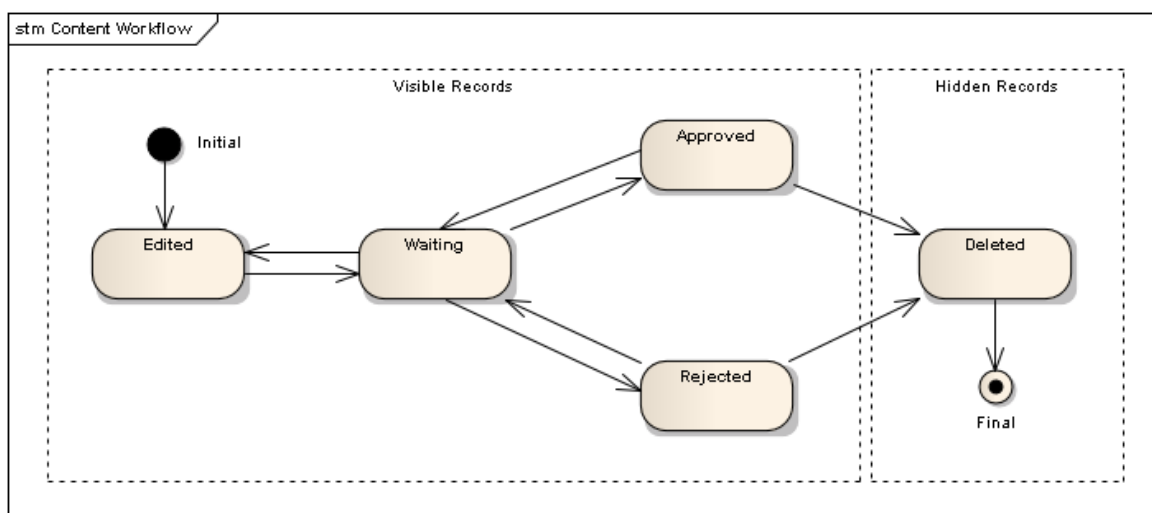


Figure 3.9: The life cycle of an element with visible and hidden work-flow states.

state	permission required	typical use
edited	edit or add	newly created and incomplete elements
waiting	edit or add	completed elements waiting for approval
approved	approve	elements published on the front-end
rejected	approve	rejected elements that require re-working

Table 3.5: Summary of work-flow states and their use.

state according to the element life-cycle. Without this permission, the user can only switch between the edited and the waiting state (assuming he has rights to edit or add the record). Table 3.5 summarizes all work-flow states with further description. Talking about the default user groups, administrators and supervisors have assigned the approve option for each content module. These users can both create and publish content. In contrast, editors do not have this option so they cannot publish anything.

3.7.2 Content Preview

Content preview is a simple feature that does exactly what the title says. Content preview allows the user to preview content elements before they are visible to the website's visitors. Preview links are available in the tree menu either for a page, a component or a selected element. In example, the component preview shows a list of articles while the element preview shows detail of a selected article. Previewed page looks exactly like the original front-end page with addition of non-approved elements and a simple panel with summary of preview details. Content preview does not support previewing changes during element editing as this information is not saved in the database before saving.

3.7.3 Personal Settings

Personal settings and options in the administration are rather simple. Basic settings include changing user's password and language for the administration. The second feature is a simple contact form. This form allows the user to directly submit a bug or contact the system's developer. Favourite links serve as bookmarks in a browser. The user can add any visible page in the administration to these bookmarks, e.g. a page, a concrete record or a quick link to the module. Added links are available both in the welcome page of the administration and in the system module Favourite. The last user-oriented option is help. Help module is planned for the future and should contain user-friendly instructions for basic tasks and operations in the system.

Chapter 4

Extending Modules

4.1 Chapter Overview

The fourth chapter describes extending modules. In contrast to the core features, each module affects only a minor part of the application, not the entire system. In the context of a content management system, these extending features are usually implemented as content modules. A particular content module is not an essential part of the system, instead it enhances its function. Urchin CMS has already included a set of basic content modules, e.g. Content module for publishing simple texts.

The first part of this chapter shortly describes several content modules coming with the default installation of the system. Content modules are used to manage and publish content on the website and adding it to the pages. The second part discusses two important concepts, these concepts are forms on the front-end and search. Full-text search is used to search for keywords in the website content and forms to collect data from users. Both topics include general overview of various approaches and a detailed solution used in Urchin CMS including corresponding content modules.

4.2 Content Modules

Concept of the component axis and modules has been already discussed in the previous chapter. Content modules are used to manage website content of various type. This section briefly describes several modules coming with the Urchin installation, their features and typical use. In addition to default capabilities, all modules might be adapted to specific requirements.

Table 4.1 shows a simple overview of content modules including a number of elements and a short description. Number of elements the module uses depends on the content module type. In the following text, all content modules are sorted alphabetically. Modules Search, QuickContact and Form will be described in two separate sections along with related concepts and general overview.

module	elements	each element represents
Articles	many	single article with perex and content
Content	one	piece of structured text
Enquiries	one	poll with question and answers
Events	many	event with dates and description
Forms	one	dynamic form with custom fields
Galleries	one	gallery with multiple images
News	many	single new
QuickContact	one	pre-defined contact form
RSS	many	single data feed
Search	zero	n/a
Sitemap	zero	n/a

Table 4.1: Overview of basic content modules.

4.2.1 Articles

Articles module allows adding multiple articles per component. In contrast to news, articles require filling in both perex and article content. Other fields are optional, e.g. date or preview images. Articles have always available both list of articles and their detail.

4.2.2 Content

Content is the simplest but most fundamental content module. Each component of this module contains only a single element with formatted text. This text is edited by a WYSIWYG editor and allows storing HTML content. Content module is widely used for text pages with formatting, images, as well as for minor text blocks, e.g. a short note, or a simple banner.

4.2.3 Enquiries

Enquiry module provides a basic tool to interact with a visitor. A component of this module has only one element that is the enquiry itself. The enquiry consists of one question and two or more answers. Number of votes is tracked for each answer. Voting attempts are logged and the module includes a simple cookie- and IP-based mechanism to prevent duplicated or fake votes.

4.2.4 Events

Events module is used for managing and displaying events. Events are actions that take place at a given time at a given venue, e.g. a conference, a football match, or an exhibition. This module allows setting date (or interval), venue, category, and description for each event. Date of beginning and short description are required fields; venue, category, and long description optional fields. On the front-end, Events module allows filtering and searching events by all fields. In projects, adjustments of this module are expected because the default version never fits all possible requirements in this area.

4.2.5 Forms

Forms module allow the user to intuitively create dynamic forms for the website. These forms can contain variable types of fields. All form fields and settings are configurable by the user in the administration. This module will be further described in section discussing the form library.

4.2.6 Galleries

Gallery works as a container for publishing photo galleries on the web. A component of this module contains only one element, that is the gallery. Each gallery consists of multiple images with thumbnails, large pictures, and short description. Gallery is available both as standard and linkable module, so it can work as a stand-alone component or be attached to other element, e.g. an article.

4.2.7 News

News is a commonly used module for publishing short news on the website. This module is similar to articles and allows multiple news per component. Required fields are perex and date of publication, optional parameters include long text and preview image. A news' detail is available only if the long text is filled in.

4.2.8 QuickContact

QuickContact module provides a basic contact form with three fields: subject, e-mail, message. Unlike dynamic forms created with Form module, quick contact form is static and immutable. This module will also be described in section discussing the form library.

4.2.9 RSS

RSS ¹ is a XML-based technology for publishing frequently changed content in a standardized format [42]. This module conforms to the RSS 2.0 specification and provides option to create feeds from website content. Each element of this module equals a single feed, each component therefore contains list of one or more feeds. Data source is chosen from active component-on-page pairs where the component must be derived from a feedable module. Feedable modules are currently News and Articles, e.g. those having textual content and multiple elements per component.

4.2.10 Search

Search module allows user to search content of the actual presentation. Details of this module are described in a separate section that concentrates on search methods and implementation in Urchin CMS.

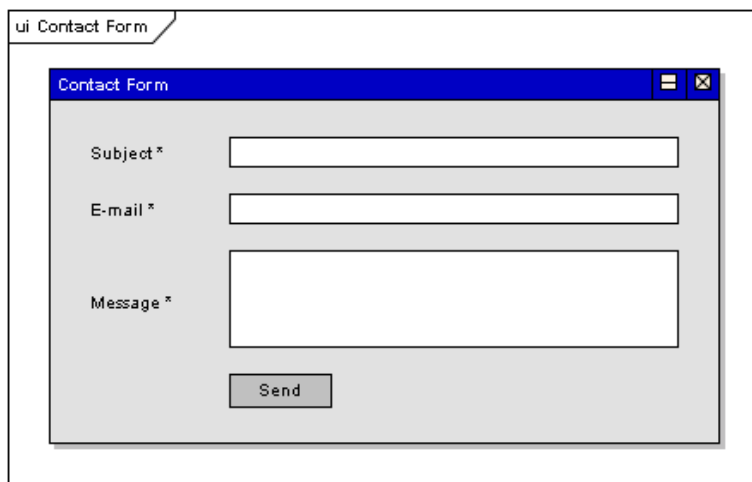
¹Rich Site Summary

4.2.11 Sitemap

Sitemap is a simple module that renders hierarchy of pages for the actual presentation. The tree of pages contains all levels of the hierarchy and includes links to all pages. This module provides a common feature that helps the visitor to navigate the website.

4.3 Dynamic Forms

Forms enable a visitor of the website to input and send data to the web application. Forms are commonly used for basic interaction with the user. Web forms look like their paper predecessors and include input elements such as text fields, radio buttons, or check boxes. Forms are defined in HTML, but require a server-side program to process data. Forms on the website are most often used for search, registration, ordering products, sending comments, or contacting a website owner. Figure 4.1 shows example of a simple contact form.



The image shows a web browser window with a title bar that says "ui Contact Form". Inside the browser, there is a window titled "Contact Form" with a blue header bar. The form contains three input fields, each with a label and an asterisk: "Subject *", "E-mail *", and "Message *". The "Message" field is a larger text area. Below the fields is a "Send" button.

Figure 4.1: Simple contact form with three mandatory fields: subject, e-mail, and message.

Web forms are declared inside a `<form>` HTML tag that defines method for its submit and includes form fields. The method is either POST or GET, as defined in the HTTP standard [19]. Forms fields provide many common graphical user interface elements. These elements are input, textarea, password, file, select, radio, checkbox, submit, and reset. However, tree views or combo boxes are not supported. Labels serve as titles connected to the fields. The following text discusses different approaches to form processing.

4.3.1 Form Implementation

In many web applications, forms are implemented from scratch. Implementation of a form consists of several steps: defining form, validating user input, handling errors, and processing data. Defining a form includes coding form and field tags, managing formatting, attaching labels, and setting up default values. Form validation requires defining mandatory fields, validating rules, and error messages. Handling errors includes redirecting back to the form,

displaying error message and pre-selecting form values. Submitted data are processed and sent to an-email or saved into database tables.

As described in the preceding text, there are many operations necessary to create even a simple form. Manual form processing is time-consuming, non-trivial, and error prone. In advanced web applications, some or all these steps are automated to speed up development and prevent errors. Urchin CMS includes an integrated library that is used exactly for these purposes.

4.3.2 The Form Library

The form library is a tool utilized for automatized form building and processing. The purpose of the form library is to simplify form definition, validating, and processing. The library does not cover additional operations, such as saving data or sending e-mails. These operations must be implemented individually. The library is usable both in the front-end and in the administration, although it is primarily intended for the front-end. In administration, crud-based generated forms are more common. Form fields are defined similarly to crud instances using controls and validators.

4.3.3 Controls

Controls are objects that encapsulate common web form fields existing in HTML, such as inputs, radios, and check boxes. In addition to rendering these elementary fields, controls enable many smart features. These features include setting and validating data format, handling default values, attaching a label, and control rendering as a part of the form. Input format is determined by the type of the control. Validators are attached to the controls and used to check the input. All controls keep values filled in if the form's submit did not succeed as well as displaying default values by the form definition.

The form library comes with many different types of controls. Basic text controls include simple input and textarea, other are used for date, time and their variants. Advanced controls are selection and multi-selection boxes, radios, check boxes. Submit controls allow sending the form, hidden control enables adding additional parameters, text control is used to display a custom text inside the form. Security controls are used to prevent duplicated submit, spam, or CSRF attacks. These controls will be discussed in a separate text. A control group object creates a group of controls that is displayed as a fieldset. Controls for uploading files are planned for the future.

4.3.4 Validators

Validators are used to check the user's input in the form. There are many types of validators that are attached to controls, each control could have assigned any number of validators. Common types of validators check if the required fields are filled in, check the input length, or match user-provided values against a pattern. Regular expression patterns are often utilized to control date, integer, e-mail, or url format. Special validators are attached to security controls to help protecting the form.

4.3.5 Form Processing

The form library is responsible for the complete process of form processing. A variant of this process is illustrated in figure 4.2. This variant is used for common one-step forms, such as those in content modules. Form settings and fields are defined in the controller and available to all views and actions in the process for rendering the form and validating its values.

Initially, the view with the form is displayed. After submitting the form, the send action is triggered. In the send action, the form is validated using defined rules. If the validation succeeds, the form is saved and the user is redirected to the feedback page. If not, the user is returned back to the view with visible error messages. Saving failure leads to the feedback page with a return link. Form values are kept after either validation or saving failure to allow review and resubmit.

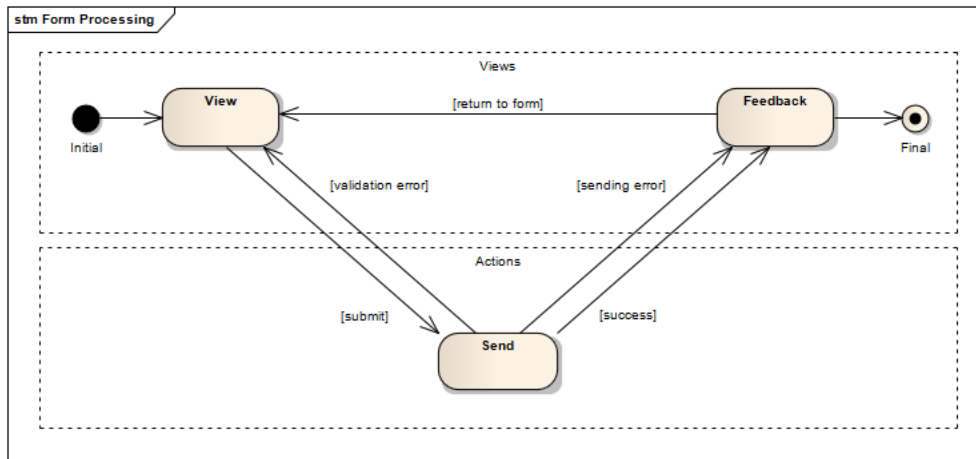


Figure 4.2: Form processing diagram with views and actions.

4.3.6 Form Security

The form library provides four complementary methods to secure web forms. The first method uses a unique token that helps preventing CSRF attacks and duplicated submit of the form. It works exactly as previously described in the [Cross-Site Request Forgery](#) section. This token-based mechanism is always present in the form, it cannot be detached because of possible security risks. Remaining three methods are implemented using controls and validators to prevent spamming and sending the form by robots. These methods are not mandatory, although strictly recommended.

The first control is simply called antispam. It requests the user to fill in a sum of two randomly generated integers. The filled number is then compared with the sum that has been saved in session. The delay control tracks the time elapsed between form displaying and its sending. If the time is lower than the defined interval, the message is considered spam. This protection relies on the fact that a human filling in the form with meaningful

form field	control	validators	notes
subject	input	not empty	
mail	input	not empty, regular expression	valid e-mail
type	selection	n/a	e.g. demand or inquiry
message	text area	not empty	

Table 4.2: Standard controls and validators used in the QuickContact module.

data cannot fill it in just few seconds like a robot. Anyway, the delay interval must be chosen very carefully.

The last type of protection is called honey pot. Honey pot control works as a logical protection. Basic idea is to enhance the form with auxiliary fields that use common names but logically do not belong to the form. These additional fields are hidden to the human visitor. In example, a form has three fields: name, surname, and address. An auxiliary field could be e.g. e-mail or phone number. The honey pot control then checks if this field is empty after the form has been submitted. Spamming robots could not distinguish these extra fields and fill them anyway. However, this method has no effect against a human spammer.

4.3.7 QuickContact Revisited

QuickContact module makes a good example of using the form library. As already mentioned, this module contains a simple contact form with several input fields. Fields are defined statically, e.g. cannot be changed. The form includes all security controls discussed in the security section and four standard controls. Table 4.2 lists all standard controls and attached validators. Form messages are sent to the e-mail and saved to the database.

4.3.8 Forms Revisited

The Forms module is used for creating custom forms in the front-end. Forms, their settings and fields are managed in the administration. This process is simple, intuitive and does not require any knowledge of programming. The user can edit form fields, recipients, text displayed after successful and failed submit, and mail content. Form fields are divided into logical groups, each with several fields. Each field has custom settings, such as adding options for radios, setting the field as mandatory, or defining allowed length or range. Incoming messages are sent via e-mail to defined recipients and logged into the database.

Form fields in the module are based upon form library controls. Each field equals a single control with one or more attached validators. Some validators are always present (format validation), use of others depends on the field settings. Table 4.3 displays controls available for this module with additional information. Grouping of fields using fieldsets is allowed, up to one level without nesting. Uploading of files is currently not supported, but it is planned for the future. Each dynamic form by default includes all security controls.

form field	control object	settings & validators
short text	input	required, length
integer	input	required, range, positive only
decimal	input	required, range, positive only, decimal places
e-mail	input	required, add to recipients
url	input	required
long text	text area	required, rows
date	date	required, interval
date & time	date time	required, interval
time	time	required, interval
list	select	required, default option
switch	radio	options, default option
yes/no	radio	default option
checkboxes	check box	options, default, checked 1+
multiple list	multi select	options, default, rows, selected 1+
displayed text	text	text content
parameter	hidden	parameter value
group of fields	control group	n/a

Table 4.3: Form fields available in the Form module.

4.4 Content Search

The coming text describes searching in the content of a website or a content management system. The first part of this section discusses two common approaches utilized for search in small- to medium-size web applications. These two approaches are internal and external search methods. Internal methods include entity-based search, content indexing, and a combination of both. External methods include search engine services and external search engines. The difference between both types is that internal search is a part of the application while external search is usually a third party service or program that is connected via a public interface. This text discusses typical use and pros and cons of every approach.

The second part describes in detail design of the search facility in Urchin CMS. The search sub-system in the application is composed of modules Index and Search and utilizes the content indexing approach to search in the website content and entity-based method to search in pages. Both parts are focused on search in the website content stored in database, searching in independent files or multimedia is not covered in this text. There are also other general search methods not discussed in this work, including inverted indexes or NoSQL databases.

4.4.1 Entity-Based Search

Entity-based search is the first of three here explained internal search methods. The idea of this principle is to provide search option for each entity independently on other entities. The searched entity is typically a single database table or a module with multiple tables.

The search is performed directly in content tables without the need for content indexing. Figure 4.3 displays a simple diagram that illustrates the entity-based search. The picture includes the querying part with three sample tables. In contrast with the content indexing method, the indexing part is not present at all.

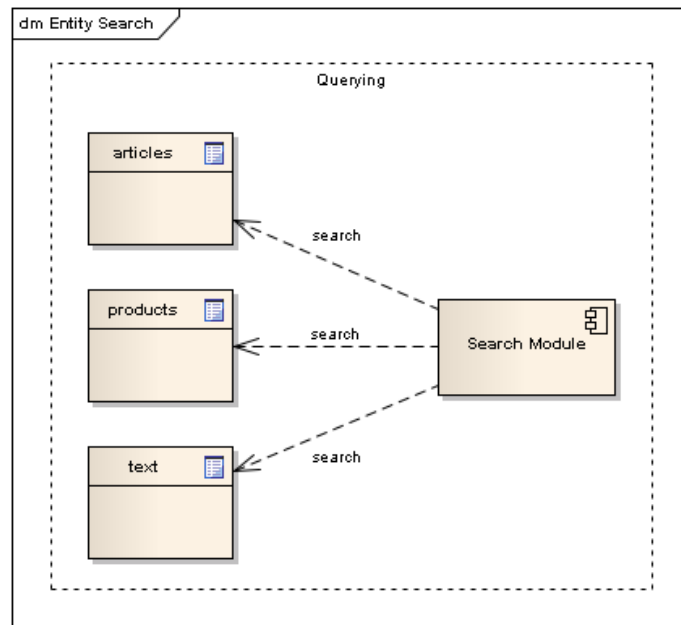


Figure 4.3: Schema of the entity-based search method.

A search process for this approach iterates all searchable tables and runs a customized query to search the keyword in each table. Found records are then displayed to the user. These records are usually first sorted by the entity (e.g. articles first, products last) and only then sorted by relevance or other criteria. This organization of results is very common for this approach.

The entity-based approach has many advantages and drawbacks. On one side, this method works well with applications that contain many diverse tables or ad-hoc structure. It is also simple to implement and enough flexible to fit individual module's requirements. On the other side, problematic areas are sorting results across entities, limited performance and presence of additional data not related to search. In example, the MySQL database currently does not support foreign keys and full-text indexes in the same table. So the developer must choose between these two often mission-critical options. If the first option is selected, the search cannot benefit from full-text indexes and always performs a full table scan.

4.4.2 Content Indexing

Content indexing is more advanced and complex method how to implement search on the website. It is hugely inspired by data warehousing and business intelligence solutions. Main

principle of this approach is to divide the search functionality into two parts, content indexing and content querying. Both parts share a database table that is not directly connected to the schema and stores data required for search. Figure 4.4 displays simple schema with both parts of the process and the indexing table. The indexing part is responsible for indexing content from the website into the shared table. The querying part provides the search itself, e.g. searches for the keyword in the indexed content.

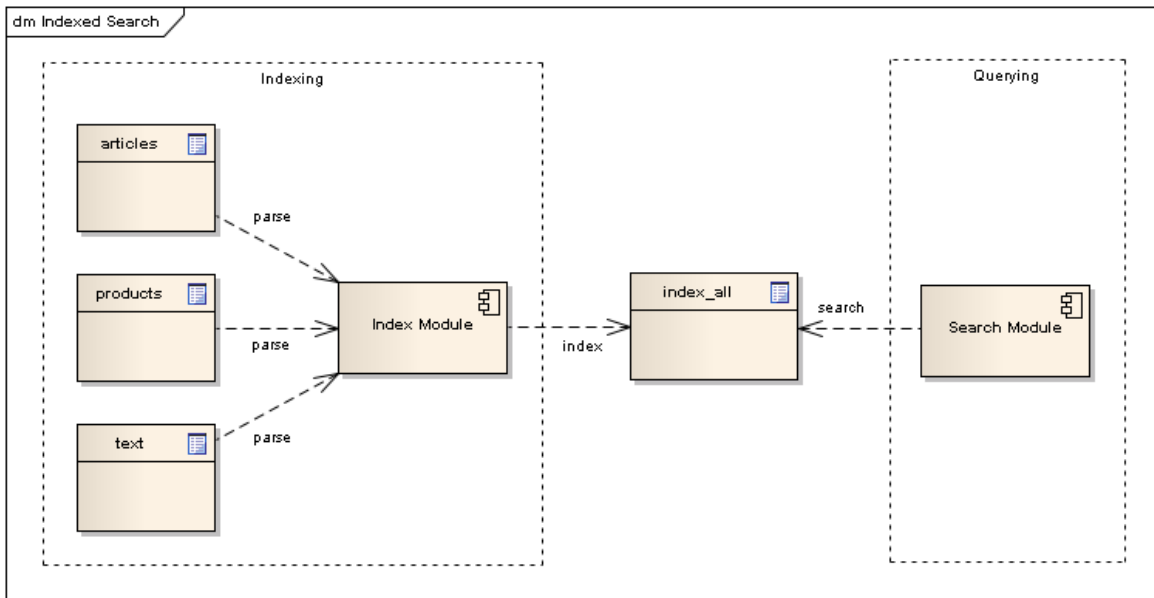


Figure 4.4: Schema of the context indexing search method.

The indexing process runs at a given time using cron or similar scheduling mechanism. The indexing interval depends on the purpose of the website, e.g. a news server requires much lower interval than a company presentation. This process also iterates all searchable tables and then indexes their content. Content indexing includes several steps: tracking changes, removing obsolete records, updating changed records and parsing new content. The querying part works similarly to the entity-based method. Instead of searching in content tables, the indexing table is searched for the keyword.

The content indexing approach has also some drawbacks and many advantages. The main disadvantage is increased complexity of this approach in comparison with the previous method. The application must be well-designed from the start to support this approach. A good example of such architecture is Urchin CMS with its concept of the component axis as will be discussed later. Other drawback is delay between content change and its indexing. The most significant advantages are related with the indexing table. The indexing table stores data in a format perfectly suitable for searching, utilizes full-text indexes for much better performance, does not contain unrelated data. It also enables trivial retrieval, sorting and filtering of found records.

4.4.3 Combined Search

The content indexing method could be combined with the entity-based approach for various reasons, e.g. if the content indexing cannot be employed for all tables or to satisfy specific requirements. There are two options for combining both approaches. The first option just complements both methods. In example, the content indexing is used for articles and news while the entity-based search works with pages. The second option is slightly different, it extends the content indexing method and uses multiple indexing tables for different purposes. An example in figure 4.5 uses two indexing tables, one table for text content and the other one for indexing e-shop categories and products.

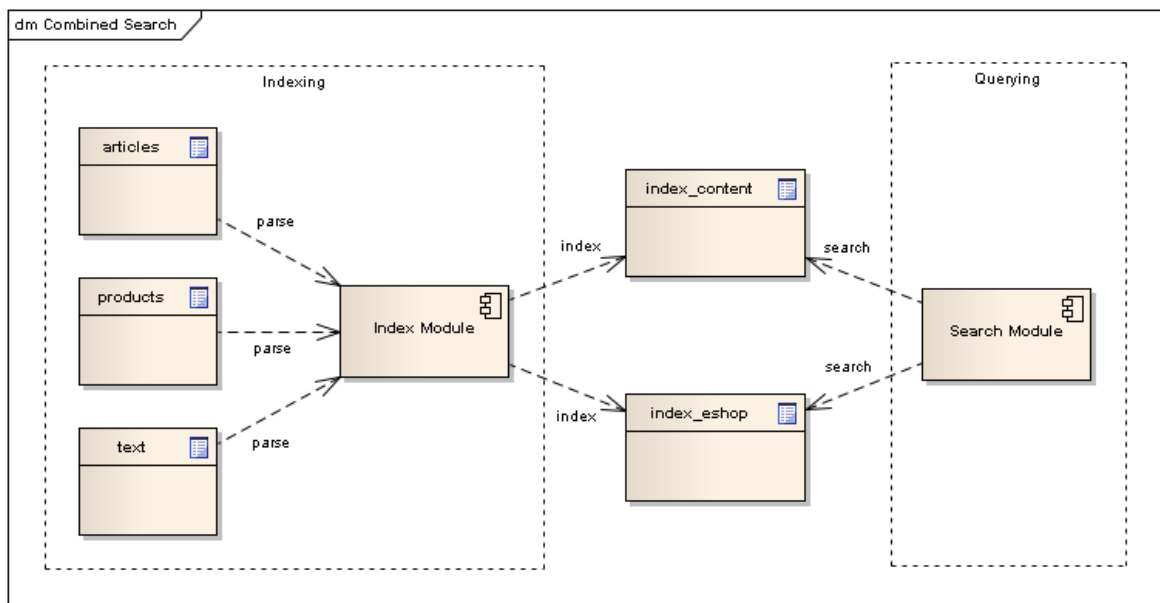


Figure 4.5: Schema of the combined search method.

4.4.4 Search Engine Service

The most simple and straightforward approach for adding an external search option to the website is to employ a third-party search engine. Many companies that maintain search engines on the internet, such as Google [13] or Bing [5], also provide search solutions for individual websites. This includes free but limited Google Custom Search [14] and Bing Box [6] for minor projects, or paid Google Site Search [15] for enterprise-level solutions. This method does not require any advanced programming knowledge. Together with usability for static websites, this approach provides a relevant solution for minor public projects. Typical drawbacks of this approach are indexing data by a third party, limited capacity or advertisement.

In example, implementing Google Custom Search includes three steps: setting up the engine, adding a search button, and creating a landing page. Setting up the search engine requires logging into the service and just clicking the create engine button. The engine

then enables basic settings and further customization. Settings include sites to search, an important setting "search only these sites", keywords, visual appearance, advertisement and other options. Two fragments of HTML code are generated after the user has finished customizing the engine. This first piece of code includes a search box, the second piece is used for displaying search results on the landing page.

4.4.5 External Search Engine

External search engines are applications that are also used for implementing advanced search on the website. Opposite to the search engine services, these programs are implemented by the website developer, not a third party. External search engines index data similarly to the content indexing approach, although they are not part of the application nor its database. Communication with the engine is realized via a public interface. Most notable open-source projects of this type are Apache Lucene [2] along with its variant Apache Solr [3], and Sphinx [43]. External search engines are highly effective, suitable for high-load projects, and able to handle any type of text files.

4.4.6 Index Module

The search facility in the Urchin application consists of two modules, Index module and Search module. Index module directly applies principle described in the general description of the content indexing approach. Figure 4.6 illustrates all important parts of the indexing process. The indexing process is triggered by the cron service at a given time once per defined interval, e.g. content is not indexed immediately after it changes. The administration also provides a simple interface to manually run the indexing. This interface is by default available only to the admin user group.

First, all indexable modules are retrieved from the table `a_module`. After that, all active components are selected for each retrieved module. Active components are those assigned to displayed pages on the front-end. The last step is parsing searchable content from elements to the indexing table entitled `a_search_data`. Obsolete records are removed from the table, changed elements are updated, and new elements added. The list of indexable modules is cached and unchanged elements are not parsed at all to reduce the load of the process.

4.4.7 Search Module

Search module is used for searching the website content within the selected presentation. This module is by default included in each installation and is composed of a search box and a result page. The search box is part of the layout and the result page is a system page, therefore it is always present and not deletable. The search process is simple and combines the indexing approach with the entity-based method. The former is used for search in the indexed content and utilizes the full-text index and options. The latter is used for simple regular expression-based search in page titles.

Figure 4.7 displays all important steps of the search process. First, the user sends a search request with desired keyword(s). Then the `a_page` table is searched for titles matching the keyword and the `a_search_data` is searched for text content containing the same keyword.

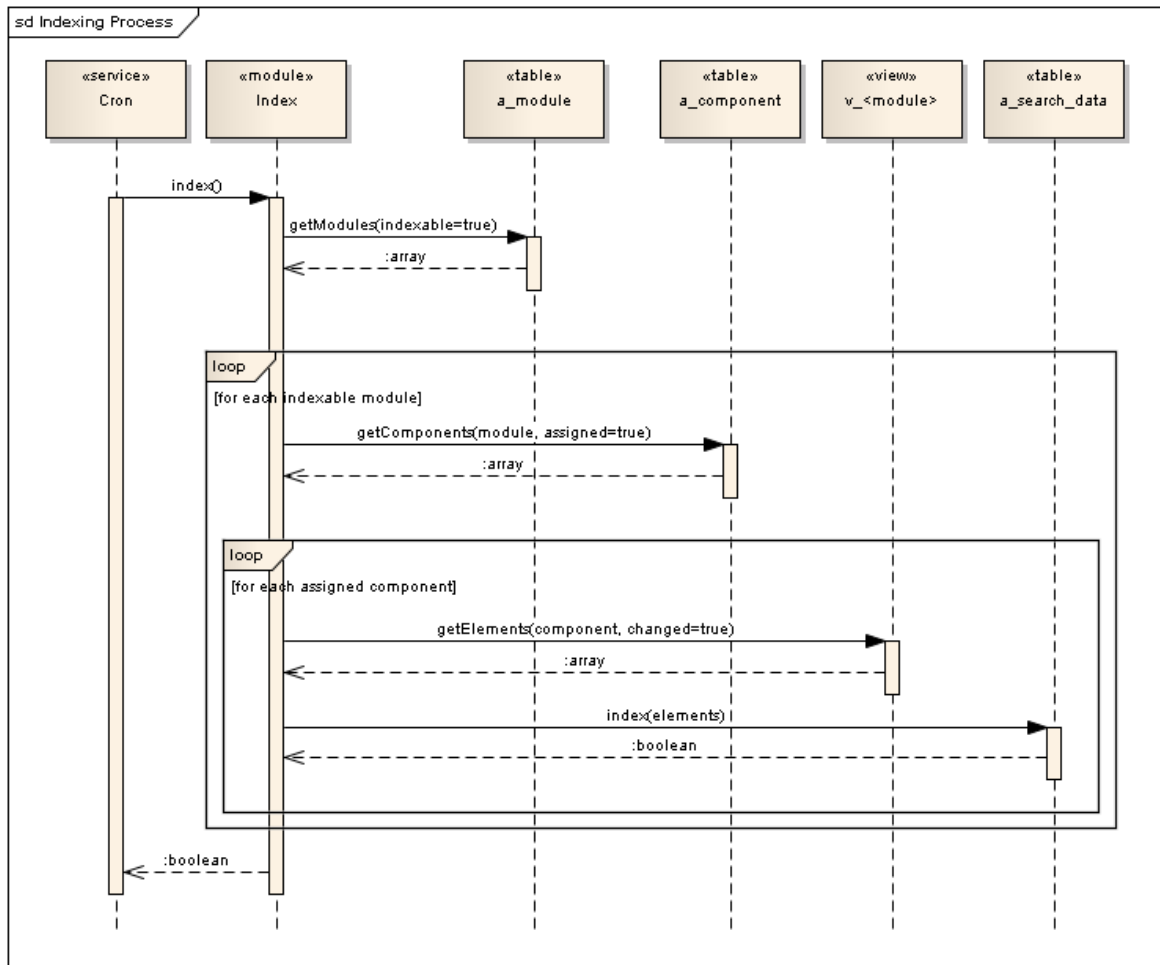


Figure 4.6: Sequence diagram for the indexing process.

Both types of records are merged and sent to the result page. Records displayed to the user are sorted by relevance and include keyword highlighting. All searched terms are logged to support basic search statistics.

4.4.8 Indexing Table

As discussed before, the indexing table is a data structure used for storing website content in a form suitable for fast and efficient full-text search. Design of this table is based on the OLAP² approach instead of the OLTP³ approach that is commonly used in transaction-dependent applications such as content management systems. For this reason, the indexing table is denormalized and not related to the application's database schema. It works similarly to a dimensional table in a data warehouse. Data in this table are extracted from other tables by the indexing service and the table itself is used solely for querying its data.

²Online Analytical Processing

³Online Transaction Processing

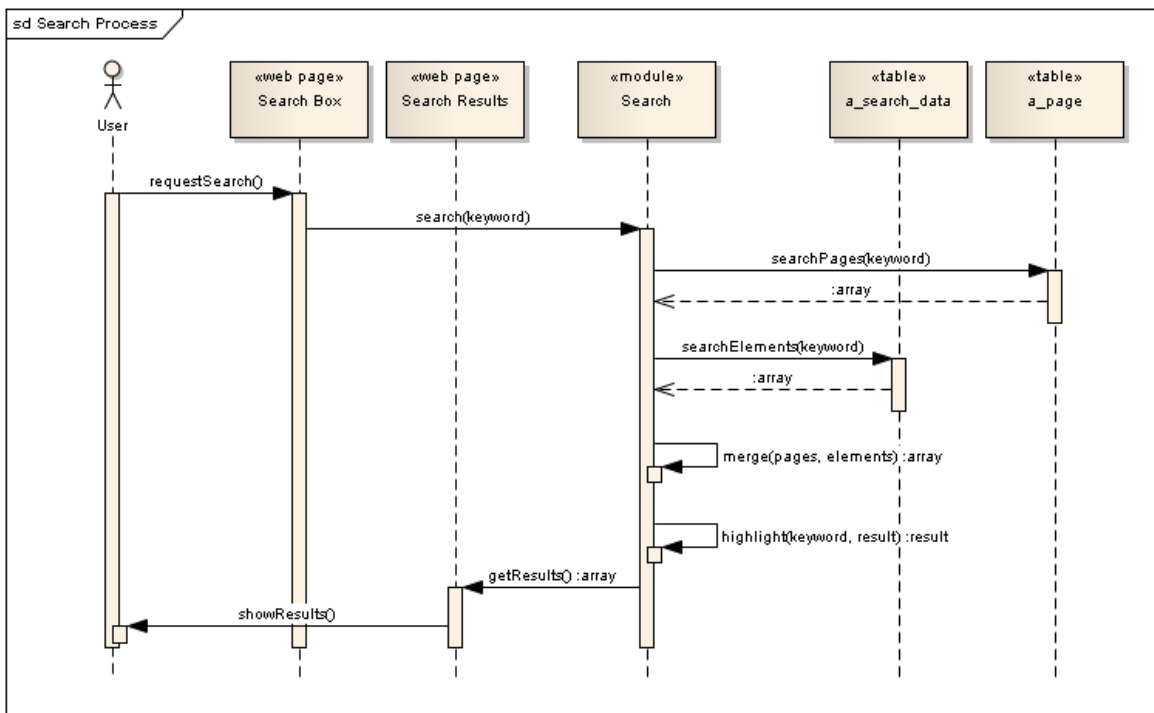


Figure 4.7: Sequence diagram for the search process.

The indexing table stores information about records, indexed text, two titles, and a timestamp. Record information include ids of presentation, module, component, and element. These ids are used to determine the presentation, join pages to the records, and quickly lookup for auxiliary data if required. Information about pages is not stored in the table, pages are joined to the records after the search has been performed. This is a trade-off between performance and complexity of the indexing table. Indexed text is used for the full-text search and component and element titles are displayed in search results. The timestamp is used for detecting changes in elements to reduce the amount of indexed elements.

Chapter 5

User Interface Design

This chapter is devoted to user interface of the administration. The first part describes visual layout of the administration as well as two specific sections, the page view and the file view. The second part briefly describes a typical layout of the front-end including important user interface elements.

5.1 Administration

5.1.1 Login Screen

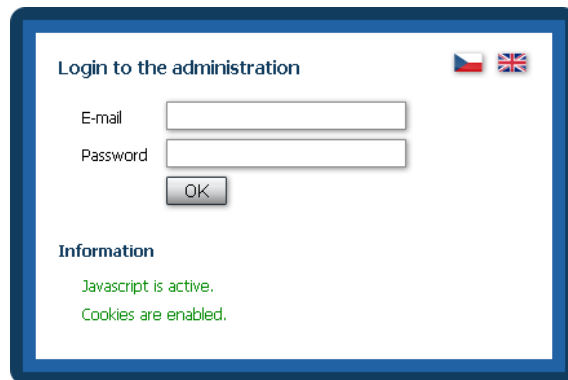


Figure 5.1: Login screen for the administration.

Login screen is the first page the user sees when accessing Urchin CMS administration. As displayed in figure 5.1, this page is simple and comprehensive. Login page is divided into three parts: login form, flags, and information panel. Login form is used for logging into the administration that requires filling in user's e-mail and password. Flags enable switching between Czech and English languages for this page. Information panel warns the user if he has disabled cookies, disabled JavaScript, or active caps lock. Cookies are mandatory, logging in with disabled cookies is not possible. JavaScript is required for comfortable use of

several user interface elements, such as interactive trees, tabs, and AJAX-based components. All fundamental functions in the administration work without JavaScript.

5.1.2 Administration Layout

User interface of the administration is designed to be both highly functional and user-friendly. The interface is primarily designed for accessing from a desktop with a standard web browser. Using a mobile device or tablet for administration is also possible. The administration supports many modern browsers¹, including older and mobile versions. Recommended resolution for effective use is 1280x800 pixels. Lower resolution makes the user experience much less comfortable. Layout of the administration is basically composed of four main panels: top, left, right, and bottom panel. Figure 5.2 illustrates these main panels using a simple wire-frame.

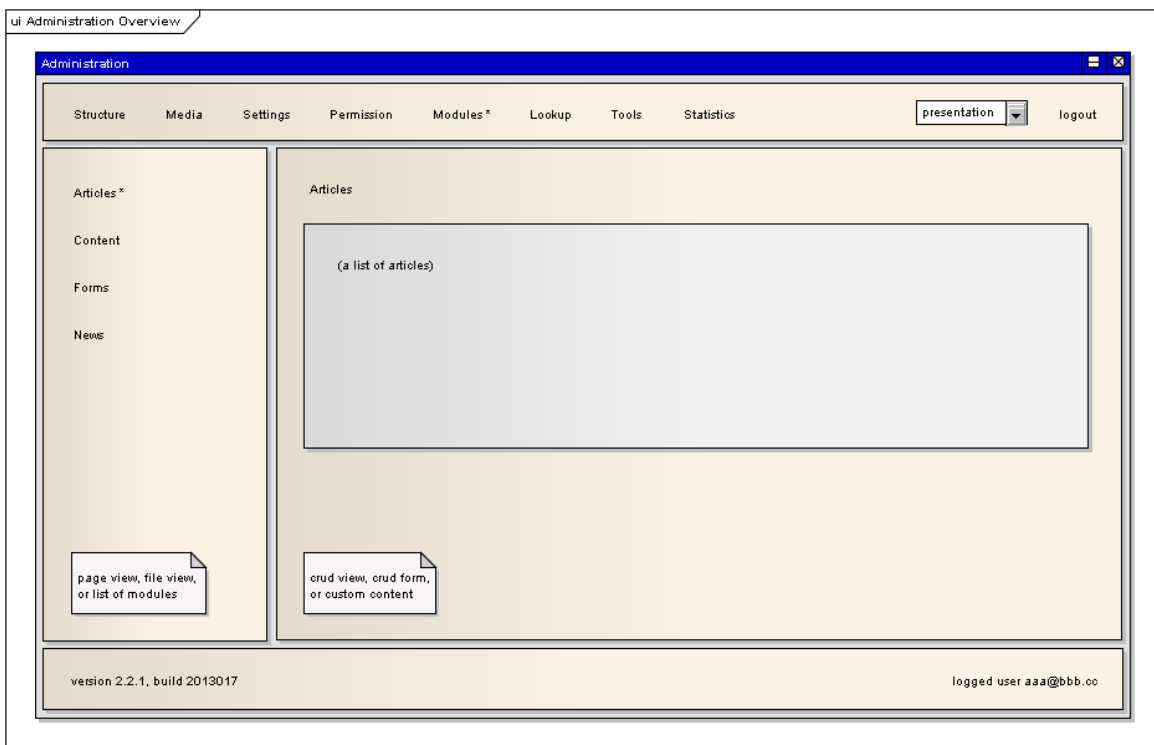


Figure 5.2: Basic layout and main panels in the administration.

Top panel is composed of main menu, presentation selector, logout link. Main menu contains links to all sections in the administration. Each section logically groups features or modules. The first link leads to the page view, the second one to the file view. Other links lead to sections that contain system and content modules. Table 5.1 lists all sections together with a short description or examples of included modules. The presentation selector is used to choose a presentation to work with. In the page view, the user works only with the selected presentation and its tree of pages. The logout link safely ends user's session.

¹Firefox, Firefox Mobile, Chrome, Internet Explorer 7+, Opera, Opera Mobile

section	short description or modules
structure	page view, component and content management
media	directory view, image and file management
settings	pages, presentations, templates, elements, personal settings
permission	users and user groups both for administration and front-end
modules	all content modules, e.g. news or forms
lookup	read-only lookup tables, e.g. states
tools	contact form, caching and search indexing
statistics	access log, search log

Table 5.1: Sections in the administration with description and modules.

Both left and right panels are configured according to the active section. The left panel usually contains a tree view or an additional menu with list of modules. The right panel contains different views, including page detail or any crud action in a crud-based module. Important sections or modules and their visual arrangement will be described in the following sections. The left half of the footer includes information about current system version and build. The half part shows e-mail of currently logged user. If the development mode is enabled, the debug panel with detailed debugging data is rendered under the footer.

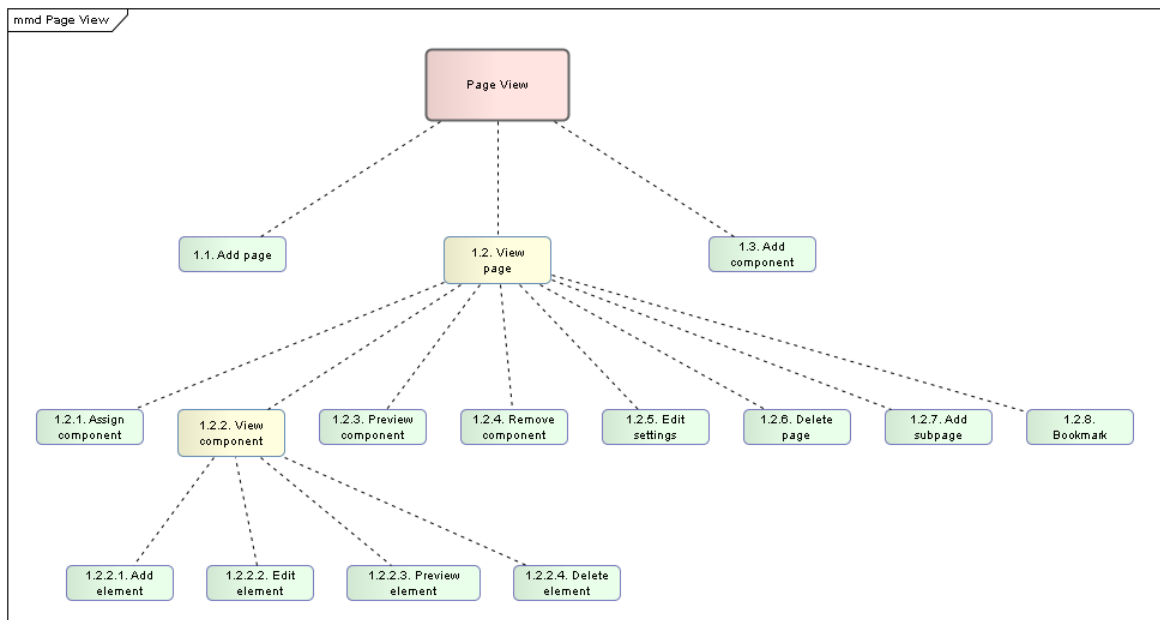


Figure 5.3: Logical organization of actions in the page view.

5.1.3 Page View

The page view is the most complex section that is used to manage pages, components and elements at one place. The left panel in the page view always contains a tree of pages in the currently selected presentation. The right panel contains either simple presentation overview or a page's details with a many additional options. Figure 5.3 displays an HTA²-like logical diagram with the most important views and actions available in the page view. The most important parts of the page view will be further discussed. Internally, this section combines functions of many modules, including application modules Page, Component, and Element, and content modules that belong to the components on the page.

The most important part of the page view is the interactive tree view with pages. The tree displays hierarchy of all pages in the selected presentation and visually reflects their position in the hierarchy. The tree is extended or collapsed by clicking it and also features a drag-n-drop function to change pages' position. Page titles are coloured black for active pages, red for pages in the client zone, and grey for inactive pages. Each page in the tree includes an on-hover tool-tip that displays several settings or flags. Available flags are summarized in the list following this text. Figure 5.4 illustrates the page view section with tree of pages and detail of the active page. The interactive tree view will be implemented in JavaScript using an open-source library jsTree [25].

- active page, inactive page
- position <number>
- displayed in the menu
- displayed in the sitemap
- redirected page
- with additional parameters

The right panel of the page view is mainly used for managing page content and settings. The default view is called positions and components and is used to manage components with content on the page. Figure 5.4 shows detail of a sample page in the right panel together with hierarchy of pages on the left side. This example displays page detail menu with additional options, a single component already attached to the page, and a component assignment panel. This menu contains several options, e.g. page settings, page preview, page deletion, or adding a new sub-page.

All occupied positions are listed under the menu together with assigned components. Each position with component includes heading, component menu, and a list of elements. Component heading contains position name, component name, and a module for this component. Component menu offers basic options for managing the component, such as preview, edit, or remove. The list of elements allows adding, editing, and deleting of elements with content. From the technical point of view, this list of elements is a specifically configured element crud instance.

²Hierarchical Task Analysis

Structure Media Settings Permission Modules Lookup Tools Statistics Presentation English Logout

English

Welcome News Services Webdesign Programming Marketing References Contact Webdesign Programming Marketing Search results Sitemap Unauthorized access Page not found

References

Positions and components Page settings On-line page preview Delete the page New sub-page

Page content [References] [Articles]

On-line component preview Component detail Edit component Copy component Remove component

Name	Perex	Content	Status	Action
cccc	Lorem ipsum dolor sit amet con...	Lorem ipsum dolor sit amet con...	Approved	
vvvv	Cras egetas lorem risus id Ve...	Eros iaculis dignissim a Quisq...	Approved	
xxxx	Sed eget non faucibus Aenean m...	Enim nulla Nullam laoreet feug...	Approved	

New record

Urchin CMS 2.2.1, build 20130407 | Josef Kunhart | © 2009-2013 Logged user: a@root.cz (sample root)

Figure 5.4: Page view with hierarchy of pages and page detail.

Structure Media Settings Permission Modules Lookup Tools Statistics Presentation English Logout

/

_gallery _wysiwyg files images

/images/

List of files New directory File upload

File name	Size	Dimension	Date of change
400_f_2136820_fbtmnrkafnhr1vts2ydfvfuwjl.jpg	3.85 kB	195 x 150	28.08.2012 14:45:47
400_f_2221631_cuw2mef3nqolapdfmfw3vfnqz.jpg	3.87 kB	200 x 150	27.08.2012 15:48:58
400_f_2382051_57is2hm0e2itaz1quh56nvr1a28n.jpg	1.94 kB	112 x 150	07.09.2012 13:34:14
400_f_2688444_77zwthaacpd0vov0datdv56l4eu1f.jpg	3.78 kB	100 x 150	27.08.2012 15:53:38
400_f_2778510_zvetaqonvsejaR246684kqhwhm8x.jpg	4.15 kB	100 x 150	07.09.2012 13:34:14
400_f_2840837_eoxhengqvbcmw678s2d8nqimfqc.jpg	5.46 kB	200 x 134	27.08.2012 15:53:38
400_f_3121484_cwvcjzuvlibwvqmfas3qetbcod6.jpg	4.15 kB	200 x 64	28.08.2012 14:45:47
400_f_3167628_vahwwo3et87et69hnpzo39kmmik.jpg	4.33 kB	100 x 150	28.08.2012 09:52:33
400_f_3456352_cqrbzovrbzrcpcqwhqvcv2dxu2.jpg	3.4 kB	200 x 134	28.08.2012 13:17:24
400_f_3556963_f4vecltvs72sv7oa1poffmobic2lo0ek.jpg	6.1 kB	114 x 150	28.08.2012 14:45:47
400_f_3703567_kvvecfwtpmim4vtxvbbhwcmbeuup.jpg	5.46 kB	150 x 150	28.08.2012 13:17:24
400_f_3708800_hrrbrvzkwin4zfo3xqgn31uxzhe7.jpg	6.34 kB	200 x 134	28.08.2012 14:44:29
400_f_3774740_pdxqumofevs6405wadufhmmmqvjl.jpg	3.62 kB	200 x 133	27.08.2012 15:48:58
400_f_3802228_lk43moanyknicwgaet5uwwfnqr36nhi.jpg	8.2 kB	200 x 134	27.08.2012 15:48:58
400_f_3844859_dtaio6tarvellairktaqvbfz55mut.jpg	7.47 kB	200 x 134	28.08.2012 14:44:29
400_f_3987441_36mzaoiokippaufav9m5a262qbiudd.jpg	3.03 kB	106 x 150	28.08.2012 09:52:32

Urchin CMS 2.2.1, build 20130407 | Josef Kunhart | © 2009-2013 Logged user: josef@kunhart.cz (sample admin)

Figure 5.5: File view with hierarchy of directories and directory detail.

Components are assigned to the page using component assignment panel in the bottom. The user can choose either to create a new component or select an existing component. The component could be assigned to any free position on the page. Each page has a limited number of positions according to its template and graphical layout.

5.1.4 File View

The file view is used for managing files and images in the system and is visually similar to the page view. This view is also composed of two main panels, as displayed in figure 5.5. The left panel represents hierarchy of directories while the right panel shows the selected directory's content. The file management is very basic at the moment. It provides simple directory listing, file upload, and simple image preview. Advanced file management, filtering, and better integration with crud components including WYSIWYG editor is planned for the coming major version.

The file view utilizes the same interactive tree as the page view, with drag-n-drop feature disabled. This section is used for managing files and images for the website. There are two types of directories in the tree, system and common directories. System directories are used for specific purpose, such as galleries or data import. Directories of this type begin are view-only and their name begins with an underscore. The user can not upload files to a system directory or its sub-folders.

Opposite to the system directories, common directories allow the user to create new sub-directories and upload files. New directories can be added up to defined level of nesting. File upload enables uploading multiple files at once. Both directories and files are not deletable to prevent errors caused by missing files. File management is independent of the database, no metadata for files are stored in the database.

5.2 Crud Layout

The crud library is used for managing most modules in sections other than the page and file views. This library has already been thoroughly presented in section [Crud Library](#), so this section focuses on short description of its user interface. A common or element crud instance includes up to four displaying actions: view, detail, edit and add. The view action displays a list of multiple records, the detail action displays a single record with possible nested instances, the edit and add actions include form for editing record or adding a new record.

Figure 5.6 shows an example of the view action that is part of an element crud instance used for managing news. This concrete instance enables all permission options and contains five records. The upper part of the view action contains add link and several filters for filtering records by defined criteria. The middle part displays basic fields of the records with detail, edit, and add links. Fields are rendered using different components including an AJAX-based component for direct editing of the name field. Most fields are sortable by clicking their headings. The lower part includes multiple delete buttons, number of records, and again the add link. Pagination is not shown due to low number of records.

Figure 5.7 displays an example of the edit action of the same crud instance as described in the previous text. Each edit form contains links to other available actions in this instance

The screenshot displays the 'News' view action in a CMS interface. The page features a navigation menu on the left with items like Articles, Content, Enquiries, Events, Categories of events, Forms, Gallery, Multupload, News, Quick contact, and Last news. The main content area shows a table of news records with columns for Name, Component, Date of publication, Perex, Status, and Action. The records are: hhhh, gggg, dddd, ssss, and aaaa. The interface includes a top bar with 'Structure', 'Media', 'Settings', 'Permission', 'Modules', 'Lookup', 'Tools', 'Statistics', 'Presentation', 'English', and 'Logout'. A footer bar shows 'Urdin CMS 2.2.1, build 20130407 | Josef Kunhart | © 2009-2013' and 'Logged user: josef@kunhart.cz (sample admin)'.

Name	Component	Date of publication	Perex	Status	Action
hhhh	News	05.03.2013 00:00:00	Quis semper Nullam Aliquam dis...	Approved	[View] [Edit] [Delete]
gggg	News	04.03.2013 00:00:00	Quis tellus at fringilla ornar...	Approved	[View] [Edit] [Delete]
dddd	News	03.03.2013 00:00:00	Est at metus id tortor tincidu...	Approved	[View] [Edit] [Delete]
ssss	News	02.03.2013 00:00:00	Lacus velit vel semper mi eget...	Approved	[View] [Edit] [Delete]
aaaa	News	01.03.2013 00:00:00	Lorem ipsum dolor sit amet con...	Approved	[View] [Edit] [Delete]

Figure 5.6: The view action of an element crud instance showing sample data.

The screenshot displays the 'News' edit action in a CMS interface. The page features a navigation menu on the left with items like Articles, Content, Enquiries, Events, Categories of events, Forms, Gallery, Multupload, News, Quick contact, and Last news. The main content area shows a form for editing a news record. The form fields include Name (hhhh), Component (News), Pretty url (hhhh), Date of publication (05.03.2013 00:00:00), Perex image, Perex (Quis semper Nullam Aliquam dis lacinia ipsum faucibus consequat turpis in. Dui nec interdum elit ligula ligula laoreet ipsum eros pretium id. Pellentesque dui pede sem Vestibulum lacinia sem adipiscing tempus condimentum ac.), and Content. The interface includes a top bar with 'Structure', 'Media', 'Settings', 'Permission', 'Modules', 'Lookup', 'Tools', 'Statistics', 'Presentation', 'English', and 'Logout'. A footer bar shows 'Urdin CMS 2.2.1, build 20130407 | Josef Kunhart | © 2009-2013' and 'Logged user: josef@kunhart.cz (sample admin)'.

Figure 5.7: The edit action of an element crud instance displaying a sample record.

at the top and a save button at the bottom. The middle part contains field labels and components. Mandatory fields are marked with an asterisk. Components vary according to the instance configuration, this concrete form uses components for editing simple texts and values, selecting an image, or WYSIWYG editing of main content. Some components contain simple tooltips with detailed description. A form used for the add action is similar to this one, with missing detail and edit buttons. As always, a concrete form's appearance depends on its configuration.

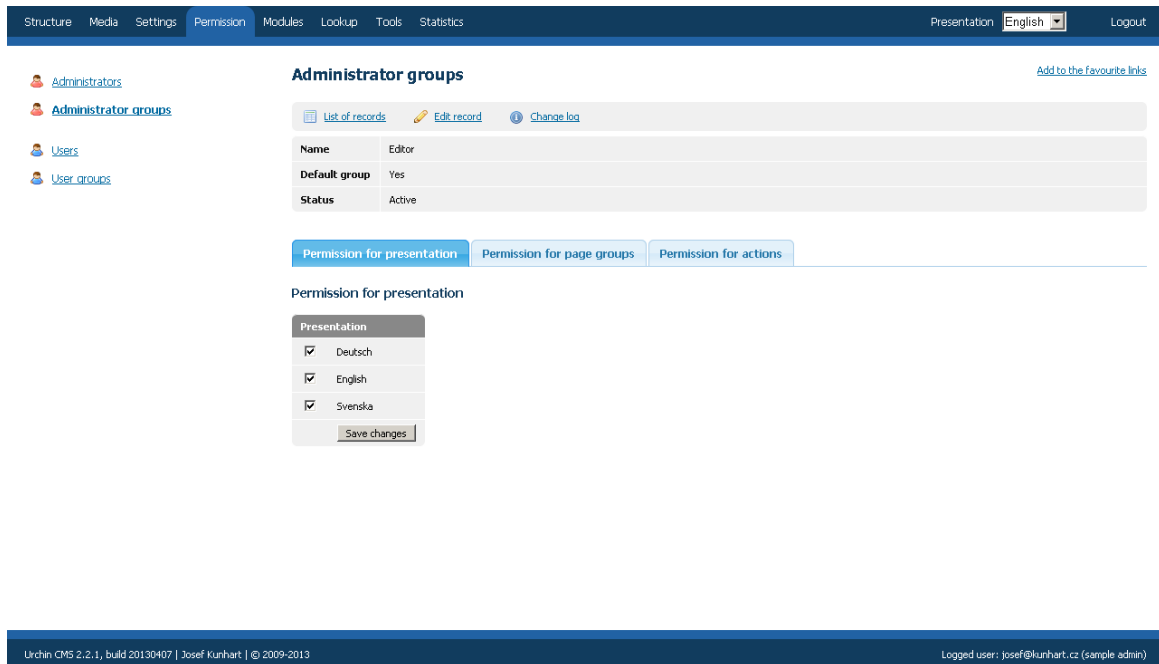


Figure 5.8: The view action of a cross crud instance nested inside a common crud instance.

Opposite to common crud instances, a cross or matrix crud instance includes only the view action. This view action both displays existing settings and allows its update. Figure 5.8 displays an example of a cross crud instance. This example shows a selected administrator group and a list of presentations that might be assigned to this group.

5.3 Front-end Layout

This section briefly describes a typical front-end page layout and its basic elements. Visual appearance of a concrete page on the website strongly depends on its graphical design, so only common features will be presented. Majority of company and commercial websites feature traditional layout with a header, menus, content, and a footer. Figure 5.5 displays a simple wireframe with example of such a layout.

A typical header contains company logo and a short textual description. Contacts, business hours, or links to social media are often part of the header. Presentation switch is used for multi-lingual websites and a search box with working search is nearly a must on

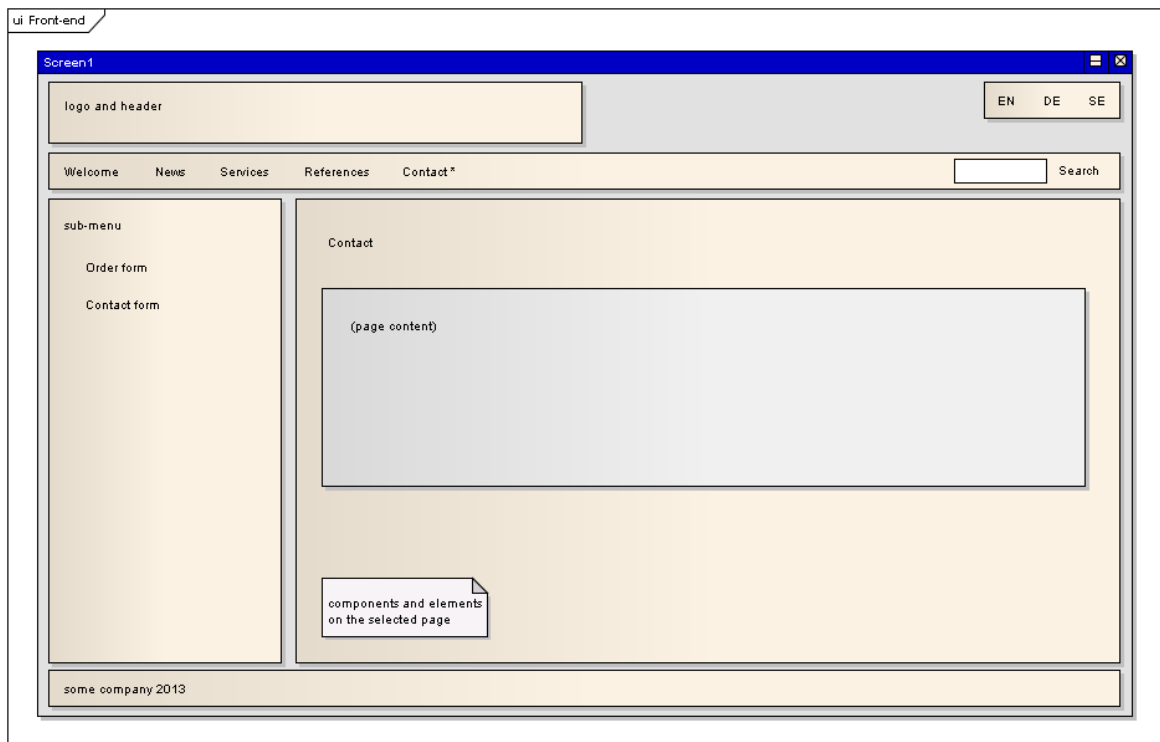


Figure 5.9: Basic layout of a front-end page.

larger websites. The main menu is located under the header and consists of links to most important top-level pages. The main menu is often horizontal, appearance of other menus depends on the graphical design. Content part differs page by page. Footer displays basic information about the company and the website author. This information includes address, contacts, and links to other websites.

Chapter 6

Testing of User Interface Design

This chapter summarizes testing of user interface design that has been performed in November 2012 as part of this thesis and subject User Interface Design [49]. This user testing was focused on user experience when working within the page view. It included several phases, e.g. screener, interviews, low fidelity testing and high fidelity testing.

6.1 Objective

Objective of this user testing was to improve existing user interface of the page tree view because several user experience difficulties had been detected in this area during development and use of the system. The page tree view is the most complex area of the administration designed for managing daily tasks like working with pages and content. The user testing took place in winter 2012 with two real users.

The testing process had three independent phases: initial phase with screener and interview, low-fidelity testing with a paper prototype, and high-fidelity testing with a real-world application. A list of ideas and improvements emerged from the testing. This list was later analysed and changes incorporated into the application. The tested version of the system was 2.0 for the low fidelity testing and 2.1 for the high fidelity testing. Corresponding updates have been implemented after the low fidelity testing in version 2.1 and after the whole testing in version 2.2.

6.2 Target Group

The target group for this project consists of both editors and common users. Editors are usually skilled in website administration and similar work. Editors typically work in a web design studio that builds websites for external clients or in a large company that maintains many own websites. Common user is a person that manages only his own personal or commercial website. A user of the Urchin CMS is expected to have some basic computer and internet skills like browsing web pages, using web forms, using e-mail, or working with a text editor. Basic knowledge of any content management system is also expected. Participants of the testing are selected using a screener and an interview.

6.3 Test Participants

6.3.1 User A

First participant is a 24-year-old male. He works in a web-design studio as an editor. His work is to code templates and manage content of various clients' websites. The aforementioned company uses its own content management system to develop web presentations and applications. He has also used another open-source system, Joomla, in her previous work.

6.3.2 User B

Second participant is a 31-year-old male. He works as a teacher in a language school. User B runs his personal website that offers translation and teaching services. He also manages his father's online portfolio. User B uses open-source Wordpress [48] to run both sites. He tried another free content management system as well.

6.4 Screener

Main purpose of a screener is to classify users and select users suitable for the interview. This screener consists of few questions connected to the project. Questions in the screener focus on basic skills required for working with a content management system.

6.4.1 Questions

A. How much are you skilled with a text processor (such as Word or Writer)?

1. unable to use
2. basic: creating and editing documents, simple formatting
3. intermediate: using styles, inserting images, managing tables
4. advanced: using macros and complex formatting, tracking changes

B. How much are you skilled with internet?

1. unable to use
2. basic: web browsing, search, e-mail use
3. intermediate: shopping online, using internet banking, editing content
4. advanced: coding templates, knowing HTML and CSS
5. professional: server- or client-side programming

C. Do you have experience with at least one content management system?

- yes
- no

6.4.2 Selection of Participants

As discussed above, test participants are selected using a screener. To qualify himself, the participant must achieve these results in the screener:

- question A (computer skill): at least basic degree
- question B (internet skill): at least intermediate degree
- question C (content management system knowledge): yes

6.4.3 Screener with User A

The first list shows screener results for User A:

- computer skill: intermediate
- internet skill: advanced
- content management system knowledge: yes

6.4.4 Screener with User B

The second list shows screener results for User B:

- computer skill: intermediate
- internet skill: intermediate
- content management system knowledge: yes

6.5 Interview

Interview is used to get basic information about participants before the testing could begin. Acquired data are analysed and used to design scenarios for both parts of the testing.

6.5.1 Topics and Questions

Topics and questions include relaxed introduction and all important topics.

A. Use of internet

- How often do you connect?
- What do you search on the internet?
- How do you get links to interesting pages?

- Do you use tabbed browsing?
- Do you bookmark pages, use download manager, additional plug-ins?

B. Content management systems

- Where do you use a content management system?
- Which system do you use in your job or for your business?
- Have you tried other CMS systems? What is your experience with them?
- Do you use help or documentation if available?

C. Managing pages

- Do you prefer using a tree view or a plain list of pages?
- Do you prefer having pages for different language versions separated?
- Do you favour complexity over simplicity or vice-versa?

D. Managing content (news, articles, texts)

- Do you use a WYSIWYG editor or prepare content outside the system?
- Do you review content before publishing it online?
- Do you like the fact of having news or articles grouped together for easier manipulation?
- What kind of feedback do you expect on an unsuccessful action?

6.5.2 Interview with User A

The following list summarizes information about the first participant, User A.

A. Use of internet

- uses internet almost every day in his job and at home
- reads mostly news, articles about web design, travelling and animals
- uses mail, social networks, chat, watches videos
- searches for links or get them from friends via social networks
- uses 10-15 tabs
- bookmarks many pages, uses web design related plug-ins in job

B. Content management systems

- currently uses CMS only in his job
- uses proprietary CMS developed by the company he works in
- had used old version of Joomla in previous work
 - experienced many problems with maintenance
 - user interface and content editing was kind of awkward and buggy
- searches for help online or asks colleagues if in his work

C. Managing pages

- usually prefers a tree view, but uses also a plain list of pages
- is familiar with a single tree with for all pages in his company's CMS
 - experiences some discomfort with that because many websites have same-named pages for multiple mutations (leads to editing errors)
- prefers complexity, likes to have most features in the page settings

D. Managing content (news, articles, texts)

- usually prepares content outside a system in HTML
- reviews content on a development server
 - this server contains almost up-to-date copy of a live website
- likes grouping of content elements, uses this feature in his job
- expects warning with detailed explanation of a problem
 - advanced feedback is better, even with more technical details

6.5.3 Interview with User B

The following list summarizes information about the second participant, User B.

A. Use of internet

- uses internet about every other day
- browses language forums, buys tickets, uses web-mail and Skype
- uses mainly search
- uses up to 5 tabs together

- has a stable number of bookmarked pages, does not change it much

B. Content management systems

- uses CMS for his and his father's websites
- uses open-source CMS Wordpress
- has installed Drupal about two years ago
 - encounters problems with resource consumption on his web hosting
- does not use help, sometimes searches for online solution

C. Managing pages

- uses only a plain list of pages
 - is content with this situation, manages only few pages
 - does not have a tree view in his system, but would like to try it
- prefers simplicity over complexity, all actions he uses should be in page settings

D. Managing content (news, articles, texts)

- uses built-in WYSIWYG editor for formatting and adding images
- reviews content just after publishing it on live site
 - likes the idea of having page preview in administration
- likes the idea of grouping news or articles together for simpler managing
- expects message with explanation
 - dislikes if anything fails and he is not informed what happened

6.5.4 Summary of Information

Analysis of user interface design of a page tree view is based upon interviews and summary of information.

- different web presentations have separated tree view
- tree view is the default method how to manage pages
- content and page settings are managed mostly in the tree view
- content preview is provided
- user can choose between WYSIWYG editor and HTML source editing
- components are used for grouping elements and assigning them to the page
- expressive and detailed feedback is provided

6.6 User Scenarios

User scenarios are used for testing and cover operations that users perform in the content management system administration. Prerequisite for each scenario is that the user is logged in the administration and he has been shortly instructed about basic concepts of the system. The basic concepts include working presentations and pages, managing content using components and elements and brief introduction to user permission. The testing environment already includes several web pages, such as welcome page and services.

Scenarios cover four fundamental operations commonly performed in the page tree view. All operations are listed after this introductory text together with detailed description of each operation. However, during the testing participants are NOT given these detailed instructions, only short description of each task. If the participant had followed the instructions step-by-step, there would be nothing to test. Instead of that, the scenarios are designed to record user experience and find potential problems with usability in the page view.

A. Add a new page for articles to English presentation

- change presentation if necessary
- choose parent page in a tree view if adding a sub-page
- click on *new page* link
- fill in the form with page information
- click on *insert* button
- fix mistakes if validation fails and try again

B. Create component for articles and assign that to the new page (added in scenario A).

- change presentation if necessary
- click on *new component* link
- select Articles module and fill in the form with component information
- click on *insert* button
- fix mistakes if validation fails and try again
- choose page in a tree view
- select free position and the newly created component
- click on *assign* button

C. Add new article to the new component (assigned in scenario B).

- change presentation if necessary
- choose page in a tree view
- navigate to the previously assigned component
- click on *new record* link
- fill in the form with article information
- click on *insert* button
- fix mistakes if validation fails and try again
- click on *preview record* link to check new article online

D. Edit existing article (created in scenario C).

- change presentation if necessary
- choose page in a tree view
- navigate to the previously assigned component and find the newly added article
- click on *record* editing link
- change information in the form as desired
- click on *update* button
- fix mistakes if validation fails and try again
- click on *preview record* link to check updated article online

6.7 Low Fidelity Testing

6.7.1 Testing Overview

Low fidelity testing focuses on testing with real users. Each testing session is an interview between a tester and a participant. The tester serves as a simple computer that manages user interface for the tested user. A paper mock-up is used for simulation of a real application and its features. The interface of the paper prototype resembles a real application, but only actions and elements important for the testing are provided. Other parts of the user interface are omitted. Participants test the prototype without previously knowing the real application.

6.7.2 Paper Prototype

Paper mock-up serves as a low fidelity prototype for further user testing. The paper prototype provides basic views, user actions and feedback for testing scenarios. Several types of paper components exist, for example tree view, menus, forms, page content and form feedback. Figure 6.1 displays all paper components used in this type of testing, including menus, main screens, basic user interface elements, and feedback messages.

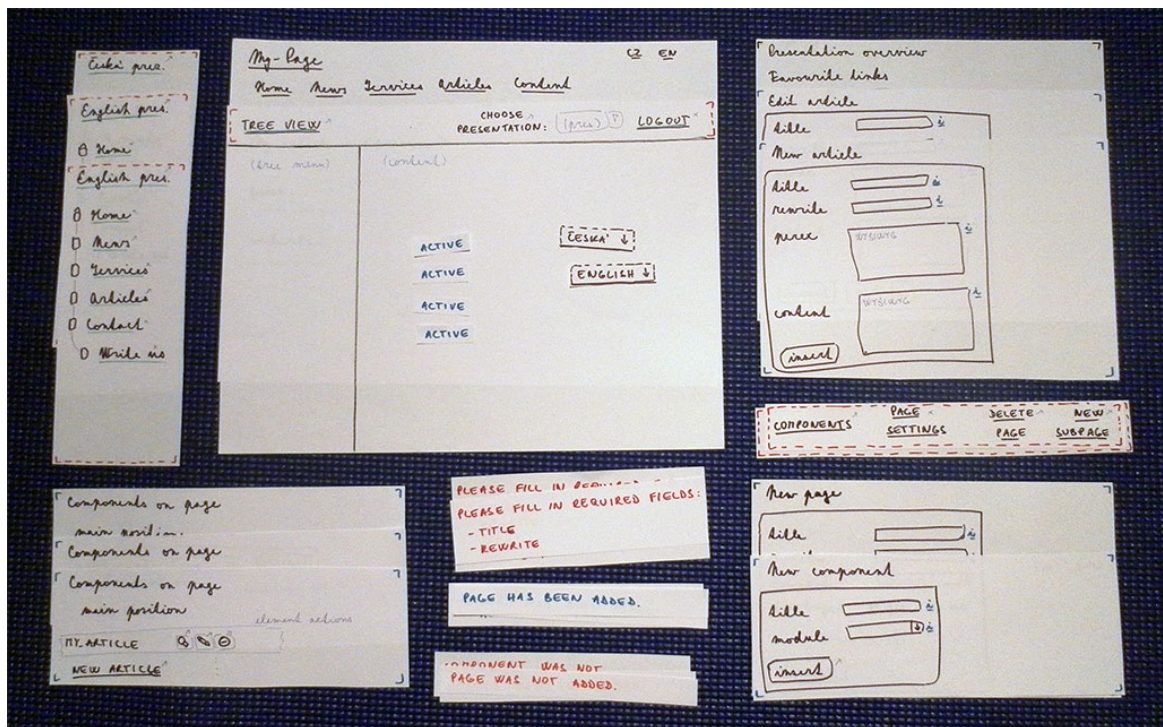


Figure 6.1: Paper mock-up overview with different types of paper components.

6.7.3 Goals and Scenarios

The goal of the low fidelity testing is to find out whether the user interface is well designed and suitable for common users. There are four scenarios that will be used for testing that have been already described and are displayed in the following list. All scenarios are linked together. User works with a page, a component, or an article created in a previous scenario.

1. Add a new page for articles to English presentation.
2. Create component for articles and assign that to the new page.
3. Add new article to the new component.
4. Edit existing article.

6.7.4 Testing Plan

- start with relaxed questions and short discussion about the initial interview
- prepare paper prototype and all tools
- tell participant basic information how the testing works
- ask for permission for recording and making photos
- explain basic features of the tested application
- test all scenarios
- discuss and review user experience

6.7.5 Testing with User B

First participant of a low fidelity testing was User B. Before the testing, meal was served and the session began with relaxed discussion about the testing and the initial interview. User B was a bit tired and curious about what to expect, so he was shortly instructed about the testing. How long does it take, what is being tested, how does it work and what is the output. Immediately after that he was told about the tested application, as it differs from his previously used content management systems. With some real-world examples, User B had no bigger problems understanding how the system works. All basic concepts such as pages, components and elements were discussed.

After this introduction, the user was shown and described in detail a paper mock-up and told about the roles of tester and user. User was asked and agreed with making photos and audio recording of the session. Due to a poor quality of the recording, only the camera was used during the session to make few photos. Then he was asked to think and speak aloud during the testing and ask for explanation or help if necessary. The scenarios were tested without showing their description to the user. Instead of giving detailed instructions to the user, he was shown scenarios and briefly told what to do.

The first scenario was adding a new empty page to the English presentation. Before starting, User B was curious about all links and features available on the prototype, so he was shortly told. Selecting a correct presentation went without any problem, same with adding a new page using the add page form. The user used help available next to the form inputs, more from curiosity than from not knowing what to do. Form validation was not triggered as the provided input did not contain any errors. Figure 6.2 shows User B after adding a new page.

Second task was to add a component to the newly created page. The user had some problems finding a new component link on the page. After the while he found the link in the presentation menu. But he expected that in the page menu, e.g. on the same place where he can select existing component. As with adding a page, the user did not have problems with filling in the form and adding a new component after he found the required link. After adding the new component, he had to assign it to the page. Third scenario was to add new article to the newly added page and component. This time, the operation was completely successful. For the last scenario, editing the new article, the user was asked to

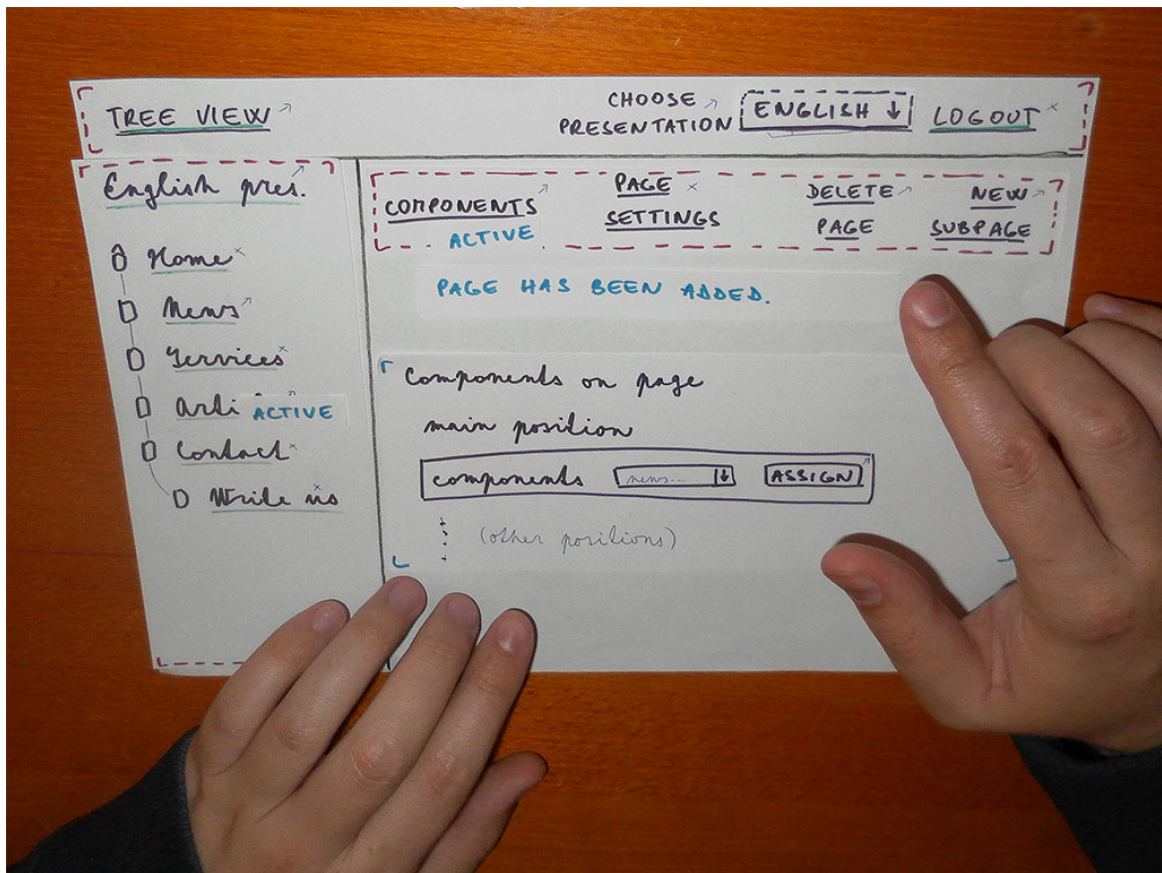


Figure 6.2: User B added a new page for articles.

work completely without any instructions. The only mistake he did was that he forgot to use a preview button for to check changes in the article online.

After the testing of scenarios, the whole testing process has been shortly discussed. The user felt comfortable with the application layout, the tree view and forms, their help and validation. He thought that component handling could be implemented better. He was also very curious about other features and functions he had seen in the prototype. For this reason, several other features were briefly described to User B after the testing was finished. Beside this output, several tips and notes for the next session were taken. The whole session took about 80 minutes, not counting the dinner and discussion on other topics.

6.8 Testing with User A

Low fidelity testing with User A took part few days after the first testing. User A works as an editor, so the things worked slightly better than with User A. This session took place in the same place as the previous testing with User B. Again, the user was first asked about the initial interview. Then he was instructed how the testing and the paper prototype works. He knows what the user testing involves, so this went really quickly as well as introduction

of the system and its features. User A was also asked about taking photos of the testing, thinking aloud and asking if necessary.

This time, a little different way of testing scenarios was chosen. User A is more experienced user of a content management system, so there was space for experimenting. He was only told basic instructions for each scenario without further brief or detailed suggestions. This tactics worked very well, as described in the coming text. All scenarios were exactly the same as in the previous testing.

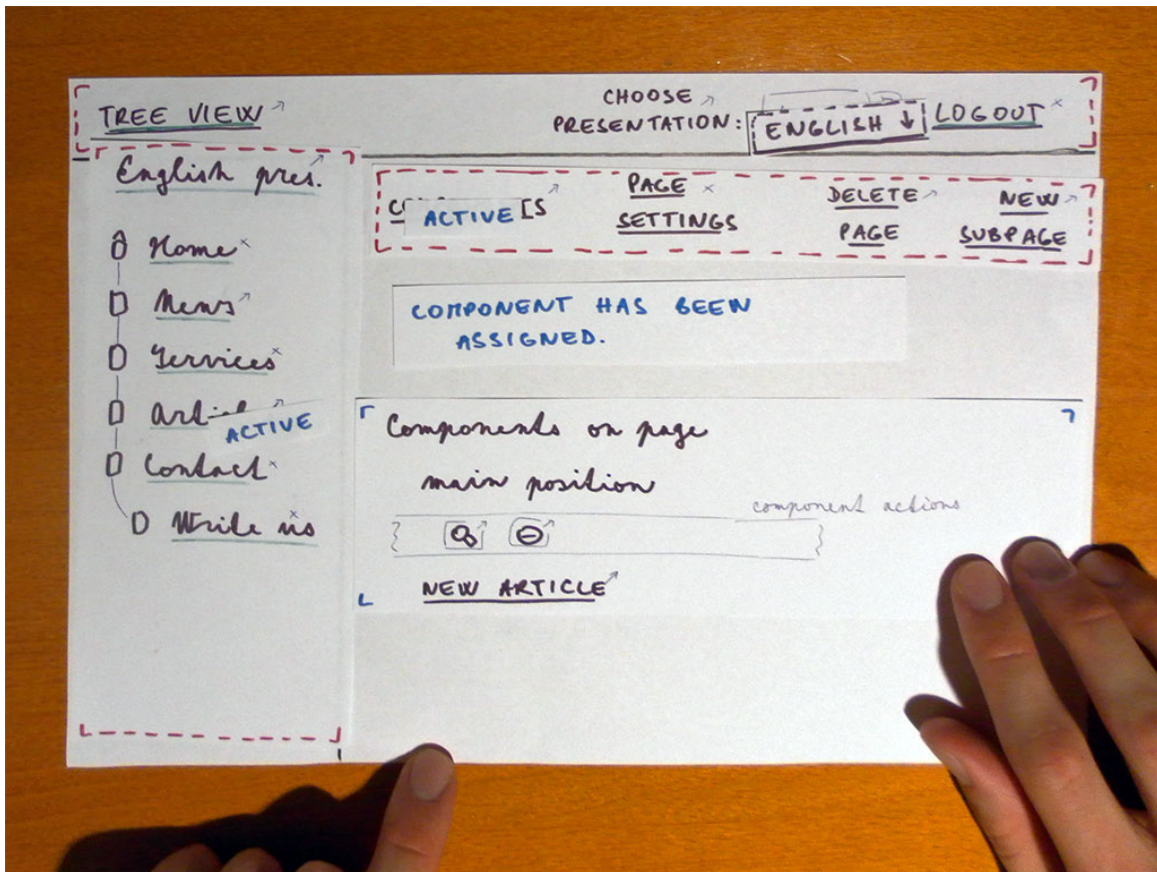


Figure 6.3: User A after adding a new component.

During the first scenario, User A did not encounter any bigger problem with the user interface. His interaction with the page tree view as well as with menus and forms was clear and straightforward. The user only added page under another page instead of directly to the presentation. This was his intention, not a consequence of a flawed interface design. The second task was again adding a new component. Unlike User B, User A had no difficulty finding a new component button. Even though, he suggested that would be great to add and assign component together in one action. And this action should be in the page menu, not the presentation menu. Figure 6.3 shows User A after adding a new component.

The last two scenarios with adding and editing an article went very well without any problems. During the testing, user was very pleased with forms and action feedback, but

a little concerned with management of components. Other features were also discussed in detail, such as layout of the user interface, menus and links. These follow common standards and work without problems in the eyes of the participant.

6.8.1 Testing Summary

These features of the user interface worked very well:

- global layout of the administration interface and menus
- main concept of presentations, pages, components and elements
- page tree view as an effective tool for page management
- brief and clear names of links and actions
- forms, their validation and help next to inputs
- straightforward and clear feedback messages
- preview feature where user can check edited content online

These features might improve:

- adding page to first versus another level could be better distinguished
- hint about the preview would be useful so the user does not forget using that

These features did not work very well:

- adding new component only from the presentation menu is a bad option
- assigning new component after its creation is confusing
- two actions for component creation and assigning should merge

Most features of the user interface worked very well. There were serious user experience problems with creation and assigning of new components. The low fidelity testing confirmed these problems, as they have been already detected before. The next part of the testing is high fidelity testing that will be performed using the existing application (with some advanced features possibly disabled). Anyway, more links, features and form options will be available. Let's see how the user experience changes with the real application.

6.9 High Fidelity Testing

6.9.1 Testing overview

High fidelity testing is a type of testing with real users that usually follows the low fidelity testing. In this concrete testing, a high fidelity prototype installed on a testing machine is used. In comparison with the paper prototype, this application has enabled all features, not only functions mandatory for the testing. The testing application uses a simple database for a small company website. Before this phase of testing, all participants have seen only a paper prototype, not the tested application itself. The tested version is 2.1 that includes many updates of the user interface since the low fidelity testing.

6.9.2 Goals and Scenarios

The main goal of the high fidelity testing is to again test if the page tree view is well designed and to gather ideas for improvements of its user interface. In contrast with the previous low fidelity testing, a real application instead of a simple paper prototype is used. All testing scenarios for high fidelity testing are exactly same as in the previous phase. See section 6.6 for details of the scenarios.

6.9.3 Preparations

First, meeting times for testing were appointed. This time, the testing was planned to take part at the author's place. Then the testing application was prepared from the actual version of the system, database was purged and new content prepared. Several copies of the testing database were made, one for author and two for both participants. After setting up the application, the author has tested all scenarios to prove that everything works correctly. A table with a laptop, a lamp, drinking water, and a comfortable chair was prepared directly before the testing session. Figure 6.4 shows testing place after setting it up.

6.9.4 Testing Plan

- start with off-topic discussion and shortly review previous testing
- prepare laptop with testing environment
- tell participant differences to the low fidelity testing
- repeat basic features of the tested system
- inform user that real application is used (with many other features)
- test all scenarios
- discuss and review user experience



Figure 6.4: Place prepared for the high fidelity testing.

6.9.5 Testing with User B

The first testing session with User B took place at author's place. Before the testing, everything was well prepared as described in section 6.9.3. First step of this session was short discussion about differences from the previous low fidelity testing. User B was told that all scenarios remain same as before and a real application is used for the testing. The fact that a real application has many more features than a simple paper prototype was also mentioned. Then few instructions about the testing laptop and environment were given. He was also asked if he wants to use a mouse or a touch pad for interaction with the tested application. User B tried briefly the touch-pad and then selected that.

As before the low fidelity testing, User B was given a short list of instructions for each scenarios. Knowing the paper prototype and the system philosophy, everything went much faster and without significant problems. Several ideas emerged from the short discussion after the testing of all scenarios. A little problem with user interface occurred, as User B had some difficulty differing the preview and detail icon when working with his article in the last two scenarios.

Icon for preview is a magnifying glass while icon for element detail is an eye. During the testing, he also denoted that presentation overview is empty and there should be some information instead of empty space. This is correct, the reason is the testing application had no favourite links or pending elements that would be displayed in this place. User B was really pleased after creating a new component because the user interface for this action has improved significantly. A large amount of interactive elements (icons, links) in the page detail imposed no problem for the user. This session took around 55 minutes.

6.9.6 Testing with User A

The second testing session with User A was arranged two days after the testing with User B. Everything was carefully prepared and then the session started. User A is more advanced user of content management systems, so the interview before giving user the testing scenarios was quite short. User A decided to directly go and work out the scenarios. He also preferred to use a mouse instead of a touch pad.

User A was given the same instructions for all scenarios as User B. He went through the testing smoothly and in addition tried some other features of the system beyond the given instructions. After the testing, various topics connected to the user interface were discussed. User A really liked the global layout of the administration, page tree flexibility and various tool tips and feedback messages throughout the administration. He denoted that adding a new component is slightly better than before. Now this operation takes only one step instead of three steps like before the improvement. He suggested few other ideas: more descriptive tool tips next to form fields, hiding other positions when editing a record (e.g. article) or disabling adding a component to redirected page as this component will never be displayed in the front-end. User A also briefly tested the work-flow system when working with articles. This session took around 45 minutes.

6.9.7 Evaluation of Testing

These features of the user interface worked very well:

- main concept of presentations, pages, components and elements
- layout of the user interface, menus and page tree view
- brief and clear links, buttons and icons
- forms, descriptive validation and feedback messages
- creating and assigning a new component in comparison with previous state
- helpful tool-tips both in the page tree and in forms

Some ideas and possible improvements:

- better icons for detail and online preview
- hide parts of the interface that are not necessary for the current action
- more descriptive tool-tips next to form fields

User experience in the high fidelity testing is slightly better than the results of the previous low fidelity testing. All tested features of the user interface worked well, although there are some details that would be improved in version 2.2 of the Urchin system.

Chapter 7

Conclusion

This short chapter discusses fulfilment of thesis objectives, future plans and ideas for development, possible new modules and licensing of this work.

7.1 Achieved Objectives

Following its main objective, this work successfully proposed architecture and design of a content management system. All fundamental concepts have been discussed in general and for the Urchin application that implements these concepts. Most important topics described in this text include database design, application architecture, pages and components, permission system, search, and user interface. Many solutions have been compared with different approaches and possibilities. In addition, user testing has been performed with intention to improve user experience with the most important part of the administration, the page tree view.

7.2 Future Plans

7.2.1 Content Versions

Content versioning is a feature that enables create and manage historical versions of website content. Old versions of the content are archived and can be restored at any time. This feature is particularly useful for tracking changes in text content or making e.g. promotional versions of products while keeping their standard versions. Content versioning works as an addition to the traditional content management. The design proposed for future versions of Urchin CMS allows managing versions on the element level, e.g. technically any element could have multiple versions. In fact, versioning will be limited to text-oriented modules and forms as an addition to the existing content management and content work-flow.

7.2.2 File Management

This part of the administration will be completely upgraded beyond simple solution presented in this work. Existing file management provides basic features, but is far from being

advanced and user-friendly. Now both file manager and components for selecting files offer a simple list of files with basic information, multi-upload of files. A first part of updates for the file manager will enable filtering and ordering files by multiple fields, image preview with thumbnails, better image resizing and possibly a trash bin. The second part includes implementing two WYSIWYG plug-ins for a user-friendly selection of files instead of manually defined path to the file. The objective here is to unify file and image selection across the administration.

7.2.3 Client Zone

The purpose of the client zone is to enable better interaction with website visitors and to provide solid ground for further improvements. The basic design of the client zone includes managing front-end users, user group, registration, login, user preferences, and logging user actions. Several features have been already implemented, such as administration of front-end user and limiting page access to registered users only. Remaining features and modules will be added in 2013, most notably the Registration and Login content modules. Many possible future areas of development will also require or benefit from the client zone, e.g. forum, newsletters or e-commerce features.

7.2.4 Technical Improvements

The most important change in this area is to replace the currently used tree model in the page tree view for another one. Current solution uses a simple parent-based solution where each node except the top-level has its parent. This tree is retrieved by a single database query, assembled in the application and cached. The future solution will use an adjacency list model. This model separates data and their relations for more efficient manipulation. Instead of having everything in a single table, one M:N table stores page relations and another table stores page data. Both described models are in detail described in [56], on pages 36-53.

Other technical improvements include separate database layer for PostgreSQL database and extending caching to content modules with text content, such as Content or Articles. Caching for content modules would require an effective solution for handling time dependencies. In example, cached records should be purged when the element begins to display or expires based on its validity interval set up in the administration.

7.3 New Modules

Several new content modules are planned for future versions of Urchin CMS. New modules will provide basic functions for a specified area while staying highly flexible and adaptable to concrete project's requirements. This approach has been already successfully applied to existing modules and helps preventing unnecessary complexity of the modules. Many new modules fall into the linked content modules category, the difference is that components of these modules are attached to elements instead of assigned to positions on a page. The following list shows examples of areas covered in future modules along with short a description. Ideas for linkable modules are marked with a star.

- login, registration, client preferences for the client zone
- a simple forum with topics, categories, and comments
- a list of most read articles, news, forum topics
- a simple catalogue of items with categories and prices
- comments, voting under an element (e.g. article or product) *
- map with coordinates and description of an element (e.g. item or shop) *

7.4 Licensing

This work is licensed under the Creative Commons BY-SA 3.0 license [7]. This type of license allows distribution of this work under the same or similar license, even for commercial use. Parts of this work licensed under this license include the text itself, all appendices, and all files available on the attached disc. This license also requires citing the author if his work is referenced.

Bibliography

- [1] *ADODB Database Abstraction Library for PHP* [online]. 2013. [cit. 25.02.2013]. Available at: <<http://adodb.sourceforge.net/>>.
- [2] *Apache Lucene* [online]. 2013. [cit. 11.03.2013]. Available at: <<http://lucene.apache.org/core/>>.
- [3] *Apache Solr* [online]. 2013. [cit. 11.03.2013]. Available at: <<http://lucene.apache.org/solr/>>.
- [4] *BBCode.org* [online]. 2013. [cit. 25.02.2013]. Available at: <<http://www.bbcode.org/>>.
- [5] *Bing* [online]. 2013. [cit. 09.03.2013]. Available at: <<http://www.bing.com/>>.
- [6] *Bing Box* [online]. 2013. [cit. 09.03.2013]. Available at: <<http://www.microsoft.com/netherlands/Web/solutions/bing-box.aspx>>.
- [7] *Creative Commons - Attribution-ShareAlike 3.0 Unported - CC BY-SA 3.0* [online]. 2013. [cit. 04.04.2013]. Available at: <<http://creativecommons.org/licenses/by-sa/3.0/>>.
- [8] *Cascading Style Sheets* [online]. 2012. [cit. 02.12.2012]. Available at: <<http://www.w3.org/Style/CSS/>>.
- [9] *Martin Fowler - P of EAA: Class Table Inheritance* [online]. 2013. [cit. 24.02.2013]. Available at: <<http://martinfowler.com/eaCatalog/classTableInheritance.html>>.
- [10] *Drupal - Open Source CMS* [online]. 2012. [cit. 02.12.2012]. Available at: <<http://drupal.org/>>.
- [11] Information Technology Laboratory: FIPS PUB 180-4, 03 2012. Secure Hash Standard.
- [12] *GNU Project - gettext* [online]. 2013. [cit. 17.02.2013]. Available at: <<http://www.gnu.org/software/gettext/>>.
- [13] *Google - About Google* [online]. 2013. [cit. 09.03.2013]. Available at: <<http://www.google.com/about/>>.
- [14] *Custom Search Engine* [online]. 2013. [cit. 09.03.2013]. Available at: <<http://www.google.com/cse/>>.

- [15] *Google Enterprise Search* [online]. 2013. [cit. 09. 03. 2013]. Available at: <http://www.google.com/enterprise/search/products_gss.html>.
- [16] *HTML 4.01 Specification* [online]. 1999. [cit. 02. 12. 2012]. Available at: <<http://www.w3.org/TR/html401/>>.
- [17] *HTML Purifier* [online]. 2013. [cit. 25. 02. 2013]. Available at: <<http://htmlpurifier.org/>>.
- [18] *HTTP - Hypertext Transfer Protocol Overview* [online]. 2013. [cit. 12. 01. 2013]. Available at: <<http://www.w3.org/Protocols/>>.
- [19] *HTTP/1.1: Method Definitions* [online]. 2013. [cit. 16. 03. 2013]. Available at: <<http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>>.
- [20] *Hibernate - JBoss Community* [online]. 2013. [cit. 06. 02. 2013]. Available at: <<http://www.hibernate.org/>>.
- [21] *MySQL 5.6 Reference Manual - 14.2 The InnoDB Storage Engine* [online]. 2013. [cit. 08. 01. 2012]. Available at: <<http://dev.mysql.com/doc/refman/5.6/en/innodb-storage-engine.html>>.
- [22] *JavaServer Faces Technology* [online]. 2013. [cit. 12. 01. 2013]. Available at: <<http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>>.
- [23] *Java EE 6 - Interface ServletContext* [online]. 2013. [cit. 12. 01. 2013]. Available at: <<http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html>>.
- [24] *Joomla!* [online]. 2012. [cit. 02. 12. 2012]. Available at: <<http://www.joomla.org/>>.
- [25] *jsTree* [online]. 2012. [cit. 20. 03. 2013]. Available at: <<http://www.jstree.com/>>.
- [26] *Martin Fowler - GUI Architectures* [online]. 2013. [cit. 10. 01. 2013]. Available at: <<http://martinfowler.com/eaDev/uiArchs.html>>.
- [27] *About NCSA Mosaic* [online]. 2012. [cit. 02. 12. 2012]. Available at: <<http://www.ncsa.illinois.edu/Projects/mosaic.html>>.
- [28] *MySQL :: The world's most popular open source database* [online]. 2012. [cit. 26. 12. 2012]. Available at: <<http://www.mysql.com/>>.
- [29] *OWASP - Top 10 2010* [online]. 2013. [cit. 25. 02. 2013]. Available at: <https://www.owasp.org/index.php/Top_10_2010-Main>.
- [30] *OWASP - Cross-Site Scripting* [online]. 2011. [cit. 26. 02. 2013]. Available at: <[https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))>.
- [31] *OWASP - SQL Injection* [online]. 2012. [cit. 26. 02. 2013]. Available at: <https://www.owasp.org/index.php/SQL_Injection>.

- [32] *OWASP - Top 10 2013* [online]. 2013. [cit. 25.02.2013]. Available at: <https://www.owasp.org/index.php/Top_10_2013-T10>.
- [33] *OWASP - Cross-Site Request Forgery* [online]. 2013. [cit. 26.02.2013]. Available at: <[https://www.owasp.org/index.php/Cross-Site_Request_Forgery_\(CSRF\)](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF))>.
- [34] *About Adobe PDF* [online]. 2013. [cit. 10.01.2013]. Available at: <<http://www.adobe.com/products/acrobat/adobepdf.html>>.
- [35] *PHP: Hypertext Preprocessor* [online]. 2012. [cit. 26.12.2012]. Available at: <<http://php.net/>>.
- [36] *PHP: Alternative syntax for control structures* [online]. 2012. [cit. 17.02.2013]. Available at: <http://www.php.net/alternative_syntax>.
- [37] *Worx International Inc. - PHPMailer* [online]. 2013. [cit. 12.01.2013]. Available at: <<http://phpmailer.worxware.com/index.php>>.
- [38] *PHP:MySQLi - Manual* [online]. 2013. [cit. 25.02.2013]. Available at: <<http://php.net/manual/en/book.mysqli.php>>.
- [39] *PHP:PDO - Manual* [online]. 2013. [cit. 25.02.2013]. Available at: <<http://php.net/manual/en/book.pdo.php>>.
- [40] *PostgreSQL: The world's most advanced open source database* [online]. 2012. [cit. 26.12.2012]. Available at: <<http://www.postgresql.org/>>.
- [41] *Qt Project - Internationalization with Qt* [online]. 2013. [cit. 17.02.2013]. Available at: <<http://qt-project.org/doc/qt-4.8/internationalization.html>>.
- [42] *RSS 2.0 Specification (version 2.0.11)* [online]. 2002. [cit. 14.03.2013]. Available at: <<http://www.rssboard.org/rss-specification>>.
- [43] *Sphinx - Open Source Search Server* [online]. 2013. [cit. 11.03.2013]. Available at: <<http://sphinxsearch.com/>>.
- [44] *TinyMCE - Home* [online]. 2013. [cit. 25.02.2013]. Available at: <<http://www.tinymce.com/>>.
- [45] *Trygve M. H. Reenskaug, personal page* [online]. 2013. [cit. 10.01.2013]. Available at: <<http://heim.ifi.uio.no/~trygver/>>.
- [46] *UTF-8 and Unicode Standards* [online]. 2013. [cit. 17.02.2013]. Available at: <<http://www.utf-8.com/>>.
- [47] *Content management system* [online]. 2012. [cit. 02.12.2012]. Available at: <http://en.wikipedia.org/wiki/Content_management_system>.
- [48] *WordPress - Blog Tool, Publishing Platform, and CMS* [online]. 2012. [cit. 28.03.2013]. Available at: <<http://wordpress.org/>>.

- [49] *XD39NUR - annotation (in Czech)* [online]. 2013. [cit. 20.03.2013]. Available at: <<http://www.fel.cvut.cz/education/bk/predmety/16/32/p1632706.html>>.
- [50] *XHTML 1.0: The Extensible HyperText Markup Language (Second Edition)* [online]. 2012. [cit. 26.12.2012]. Available at: <<http://www.w3.org/TR/xhtml1/>>.
- [51] *Extensible Markup Language (XML)* [online]. 2013. [cit. 10.01.2013]. Available at: <<http://www.w3.org/XML/>>.
- [52] *JQuery: The Write Less, Do More, JavaScript Library* [online]. 2012. [cit. 26.12.2012]. Available at: <<http://jquery.com/>>.
- [53] BUSCHMANN, R. R. H. S. P. S. M. F. M. *Pattern-Oriented Software Architecture Vol 1: A System of Patterns*. 1. John Wiley and Sons, 1996. ISBN 978-0471958697.
- [54] FOWLER, M. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002. ISBN 978-0321127426.
- [55] KARAGKASIDIS, A. Developing GUI Applications: Architectural Patterns Revisited. In *EuroPLoP 2008: 13th Annual European Conference on Pattern Languages of Programming*, july 2008.
- [56] KARWIN, B. *SQL Antipatterns: Avoiding the Pitfalls of Database Programming*. Raleigh, North Carolina, 2010. ISBN 978-1934356555.

Appendix A

List of Abbreviations

- AJAX** Asynchronous JavaScript and XML
- API** Application Programming Interface
- CMS** Content Management System
- CRM** Customer Relationship Management
- CRUD** Create, Read, Update, Delete
- CSRF** Cross-Site Request Forgery
- CSS** Cascading Style Sheets
- GUI** Graphical User Interface
- HMVC** Hierarchical Model-View-Controller
- HTML** HyperText Mark-up Language
- HTA** Hierarchical Task Analysis
- HTTP** Hypertext Transfer Protocol
- Java EE** Java Platform, Enterprise Edition
- LAMP** Linux Apache MySQL PHP
- MVC** Model-View-Controller
- OLAP** Online Analytical Processing
- OLTP** Online Transaction Processing
- ORM** Object-Relational Mapping
- OWASP** The Open Web Application Security Project
- PAC** Presentation-Abstraction-Control

PDF	Portable Document Format
PDO	PHP Data Object
PHP	PHP: Hypertext Preprocessor
RSS	Rich Site Summary
SEO	Search Engine Optimization
SQL	Structured Query Language
SSL	Secure Sockets Layer
UCMS	Urchin Content Management System
URL	Uniform Resource Locator
WAMP	Windows Apache MySQL PHP
WCF	Web Component Framework
WCFS	Web Component Framework: Simple
WYSIWYG	What You See Is What You Get
WWW	World Wide Web
XML	eXtensible Mark-up Language
XSLT	eXtensible Stylesheet Language Transformations
XSS	Cross-Site Scripting

Appendix B

List of Terms

B.1 Architecture

Common Crud A crud-based controller is used to manage records in a single main database table in the administration. The common crud employs components, validators and similar objects to work with records.

Component (crud) An object in the crud library that encapsulates form field and is responsible for its visual appearance and formatting.

Converter An object used for converting values to strings and strings to values.

The Crud Library A built-in library that enables rapid and flexible development of the user interface in the administration. Common settings and elements are defined instead of programming.

Decorator An object that visually enhances displayed data independently of the component.

Element Crud A special variant of the common crud used to work with elements.

Entry A container used in the crud library that holds a single component and validators.

Filter A simple object used for filtering records in the crud view.

Front Controller The single access part to the application that routes incoming requests to controllers and is also responsible for access control and localization.

Hierarchical Model-View-Controller An extended version of the MVC pattern An application following this architectural pattern consists of multiple hierarchically organised MVC triplets.

Model-View-Controller An architectural pattern used to divide a web application into three separated layers, the model, the view, and the controller part.

Validator (crud) An object used for validating user input inside the entry.

B.2 Core Features and Modules

Action A basic operation in the administration. Actions employ options to control user access.

Application Module A type of module that is responsible for managing basic structure of the website or cooperate with element modules. Application modules are used both in the front-end and administration.

Component (concept) A group of elements managed together as a single item. Component is derived from a module and assigned to the concrete position on a page. Each component might be assigned to multiple pages, even across the presentations.

The Component Axis A basic concept dealing with the horizontal partitioning of the website. The website content is divided into module-based components that are assigned to the pages.

Content A common term that generalizes both static and dynamic features of a website. Content could be e.g. text, articles, forms, images, or multimedia.

Element A basic low-level unit of a website. An element corresponds to a basic low-level unit of content, such as a single article, a web form, or a simple static text.

Element Module A type of module that is employed to manage and deliver content to the front-end. Element modules work with special database records called elements.

Linkable Module A special type of module. A component derived from a linkable module is assigned to another element instead of to the position.

Module A software package that extends the Urchin application. Modules usually consist of database tables, classes, templates and a default set of permission. Modules serve as templates for creating components.

Option An access right assigned to the module and used to check access to actions. The set of options is fixed, available options are: view, detail, edit, add, delete, and approve.

Page A basic part of the website. Each page has assigned a single template with fixed number of positions.

The Page Axis A basic concept dealing with the vertical structuring of the website. The website is composed of presentations, each presentation has a tree of pages with content.

Position A slot on a page used for assigning components with diverse content.

Presentation A top-level part of a website. Each presentation has assigned language and consists of multiple pages arranged in a tree structure.

System Module A type of module that is used in the administration to handle internal settings and tasks.

System Page A special type of page that ensures fundamental functions and is always present in the presentation.

Template A structure that defines layout of a page. A template is closely connected to the graphical design of the website and has assigned a fixed number of positions.

Website A set of related pages with content accessible via Internet or another network. A typical website is divided into one or multiple presentations.

B.3 Extending Modules

Control An object in the form library that encapsulates form field and handles its validation, formatting and additional settings.

Form Common element of the user interface used for gathering information from the visitor.

Indexing The process of extracting data from website into the indexing table or tables.

Indexing Table A special table used for storing indexed content.

Search Engine A software used for indexing data from the world wide web.

Validator (forms) An object used for validating user input inside the control.

B.4 User Interface and Testing

Interview A part of a user testing used to retrieve basic information about participants to analyse and prepare testing scenarios.

High Fidelity Testing A type of testing with users that utilizes an advanced electronic prototype or uses a real application.

Low Fidelity Testing A type of testing with users that utilizes a paper or similar simple prototype of a developed application.

Screenner A test used to determine if the participant is suitable for a particular testing.

User Interface A visual part of the application that is used for interaction between the user and a software system.

Appendix C

List of Content Modules

C.1 Basic Modules

The first list displays default content modules available in the Urchin application.

Article basic articles

Content text content with HTML formatting

Enquiry enquiries with questions and answers

Event events with calendar, categories and venues

Form a dynamic form builder with various fields and validation

Gallery** simple photo galleries

LastArticles a list of last articles

LastEvents a list of last events

LastNews a list of last news

News basic news

QuickContact a contact form with subject, e-mail and message

RSS RSS feeds, use other components as a data source

Search full-text search in the actual presentation

SendLink* form that sends link to the actual page

Sitemap a tree view of pages in the current presentation

C.2 New Modules

The second list shows ideas and proposals for new modules. Not all ideas will be realized.

Banner banner and advertisement management, statistics for banners

Client user settings and preferences in the client zone

Comment* form for posting user comments

Forum discussion forum with topics, threads and posts

GoogleMap* a simple map attached to the element

Login login form to the client zone

Newsletter e-mailing to the registered users

Product product management with properties and price management

Registration registration form to the client zone

Top<Element> a list of most visited elements, e.g. articles, news, products

Vote* voting for an element

* Denotes a linkable (only) module.

** Gallery module is available both as standard and linkable module.

Appendix D

Electronic Disc

This thesis also includes a disc with electronic version of this work as required by the conditions for submitting theses. The following list presents directory structure of the disc with short description of its contents.

- /diagrams/ - project for Enterprise Architect with all diagrams
- /figures/ - all images and diagrams used in this text
- /thesis/ - electronic version of the thesis
 - /final/ - final version in PDF
 - /source/ - source code for LaTeX

Appendix E

Database Model

This appendix contains conceptual database models of the Urchin application. Both images are stored on the electronic disc attached to this work.

E.1 Core Application

The first diagram displays database tables of the core application and their relations. All core tables are prefixed a_* after the first letter of the alphabet. The picture shows three main areas already described in the text, the page axis, the component axis, and the permission system together with several interjecting tables. For the reasons of clarity and simplicity, tables with translated terms are not included. These tables contain translations of static terms, such as module name or page states. The path to this diagram is:

- /figures/urchin-db-cms.png

E.2 Content Modules

The second diagram shows database structure of all basic content modules. This set of content modules is always available in the Urchin installation. On the database level, each module consists of one main table that inherits a_element table, one view, and zero or more additional tables. Module table are prefixed m_*, views are prefixed v_*. Relations to the system modules are also shown if present. Basic modules Search and Sitemap do not require any database tables and are not shown in the schema. The path to this file is:

- /figures/urchin-db-modules.png

Appendix F

Sample Application

This appendix presents a sample web application that is designed using Urchin content management system. The sample application serves as a presentation of a fictional internet agency called "Some Company" that offers services in multiple European countries. The following text describes front-end structure and content of the website.

F.1 Description

F.2 Website and Presentations

Website consists of three different presentations that serve as language versions. Every presentation therefore uses a different language. The list of presentation follows.

- English presentation
- Swedish presentation
- German presentation

F.3 Templates and Positions

The website utilizes three different types of pages. The type of a concrete page is determined by template it has assigned. Available templates are home page, common page, and wide page. Each template is connected to a different graphical layout and contains a different fixed number of positions. The following list shortly describes all templates while figure [F.1](#) depicts layout of these templates including header, menus, and positions.

1. home page template
 - specific graphical layout, different to other templates
 - three positions: left, middle, right

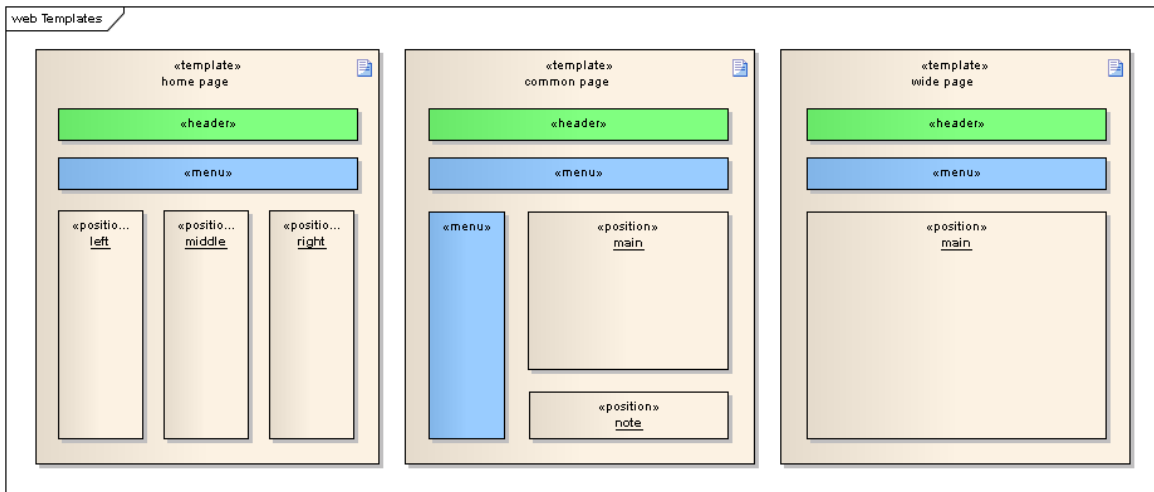


Figure F.1: Templates available in the sample application.

2. common page template

- standard graphical layout with the left menu
- two positions: left, note

3. wide page template

- standard graphical layout without the left menu
- one position: main

F.4 Hierarchy of Pages

Each presentation in the website consists of multiple pages with website content. Page hierarchy is independent for each presentation. Table F.1 describes details of each page, such as assigned template and basic settings. Figure F.2 under this list explains logical structure of the website with details for the English presentation. Basically, only the welcome page employs the home page template. Pages for services and contacts use the common page template and pages without sub-pages or parent pages has assigned the wide template. System pages are not displayed in the picture except for search results and sitemap.

F.5 Modules and Components

All presentations contain multiple types of content, such as simple text, news, forms, or references. Components are used to display content on the front-end. Table F.2 shows components assigned in the same English presentation as described before. The table includes a short description of each component and module that identifies the component. Empty positions are not shown in the table.

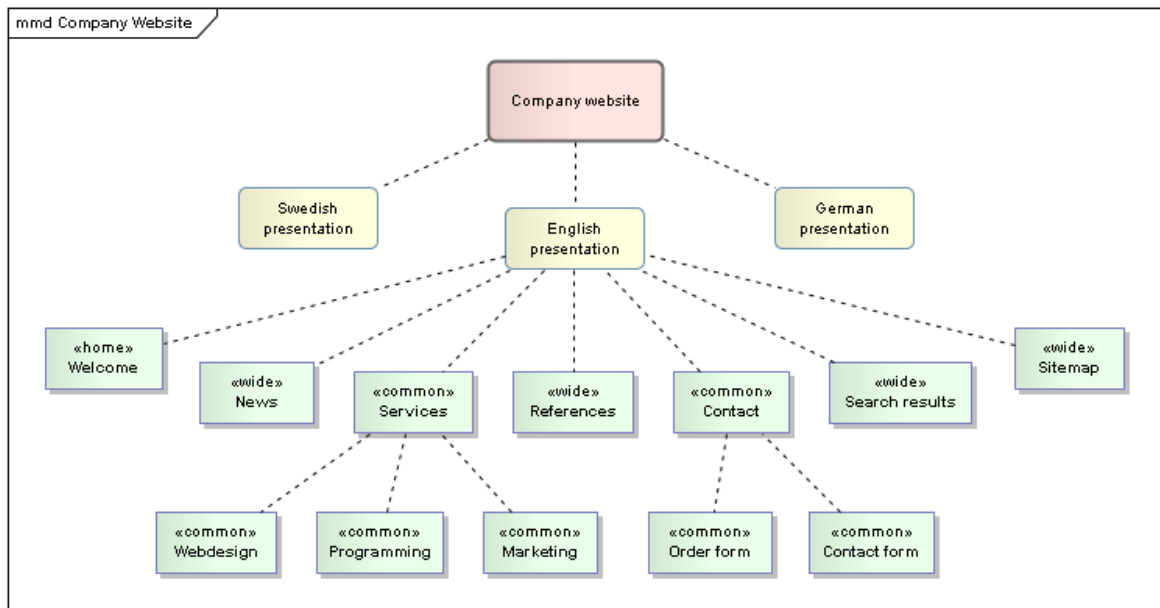


Figure F.2: Logical hierarchy of the sample website.

page	template	settings and parameters
Welcome	home page	is_homepage = 1
News	wide page	n/a
Services	common page	no content, redirected to the first sub-page
Webdesign	common page	n/a
Programming	common page	n/a
Marketing	common page	n/a
References	wide page	n/a
Contact	common page	n/a
Order form	common page	n/a
Contact form	common page	n/a
Search results	wide page	system page, not in menu, is_fulltext = 1
Sitemap	wide page	system page, not in menu, is_sitemap = 1

Table F.1: Templates and settings of pages in the English presentation.

page	position	module	component
Welcome	left	Content	introduction and basic information
	middle	Content	teaser for all services
	right	LatestNews	the last two news
News	left	News	news about the company and its projects
Services	n/a	n/a	n/a
Webdesign	main	Content	web design offer and description
	note	Content	note 'prices including tax'
Programming	main	Content	programming offer and description
	note	Content	note 'prices including tax'
Marketing	main	Content	marketing offer and description
	note	Content	note 'prices including tax'
References	main	Articles	information about completed projects
	n/a	Gallery	photo gallery for each reference
Contact	main	Content	address, contacts to salesmen
Order form	main	Form	dynamic form with individual fields
Contact form	main	QuickContact	pre-defined contact form
Search results	main	Search	full-text search in current presentation
Sitemap	main	Sitemap	tree of pages with links

Table F.2: Components assigned to pages of the English presentation.