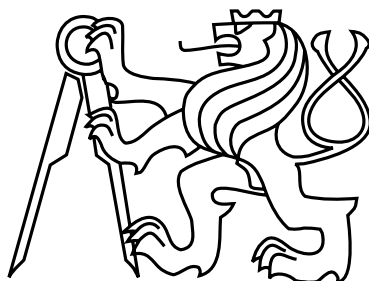


České vysoké učení technické v Praze
Fakulta elektrotechnická



Bakalářská práce

Vývoj redakčního systému NORS

Daniel Milde

Vedoucí práce: Ing. Martin Komárek

Studijní program: Softwarové technologie a management

Obor: Softwarové inženýrství

květen 2009

Poděkování

Chtěl bych poděkovat vedoucímu práce, Ing. Martinu Komárkovi, za podnětné připomínky, kterými směřoval vývoj této práce.

Dále bych chtěl poděkovat své přítelkyni za bezbřehou trpělivost a nakonec své rodině za podporu během studia.

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v příloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 8.6. 2009

.....

Abstract

The work documents development of web content management system for platform PHP5/MySQL. The reason for birth of this project was the absence of compact CMS which had code separated into tiers and adhered to object paradigm. That's why is major attention paid to design. The final application is build on the top of Model-View-Controller architecture. It strongly uses design patterns, has very simple installation and configuration and is easily extensible. Application has surprisingly a higher performance than widespread non-object CMSs and uses relatively little memory.

Abstrakt

Práce dokumentuje vývoj webového redakčního systému (WCMS) pro platformu PHP5/MySQL. Důvodem pro vznik projektu byla neexistence kompaktního CMS, který by měl kód rozdělen do více vrstev a dodržoval objektové paradigma. Velká pozornost je proto věnována návrhu. Výsledná aplikace je postavena na architektuře Model-View-Controller, silně využívá návrhové vzory, má velmi snadnou instalaci a konfiguraci a je lehce rozšiřitelná. Aplikace je překvapivě výkonnější než rozšířené neobjektové redakční systémy a využívá poměrně málo operační paměti.

Obsah

Seznam obrázků	xv
Seznam tabulek	xvii
1 Úvod	1
1.1 Zadání bakalářské práce	1
1.2 Úvod do problematiky	1
2 Popis problému, specifikace cíle	3
2.1 Cíl práce	3
2.2 Cíl projektu NORS	3
2.3 Důvody vzniku	3
2.4 Cílová skupina produktu	3
2.5 Teze projektu	3
2.6 Rešerše existujících projektů	4
2.6.1 Metodika testu	4
2.6.2 Srovnávané projekty	4
2.6.2.1 Drupal	4
2.6.2.2 Joomla!	4
2.6.2.3 Nucleus	5
2.6.2.4 PHP-Nuke	5
2.6.2.5 phpRS	6
2.6.2.6 RS2	6
2.6.2.7 Typo3	6
2.6.2.8 Wordpress	7
3 Požadavky a jejich analýza	9
3.1 Požadavky	9
3.1.1 Funkční požadavky	9
3.1.1.1 Back-end	9
3.1.1.2 Front-end	9
3.1.2 Obecné požadavky	9
3.2 Analýza	10
3.2.1 Doménový model	10
3.2.2 Stavové diagramy	11
4 Návrh řešení	13
4.1 Platforma	13
4.1.1 Nároky na konfiguraci	13
4.1.2 Framework	13
4.2 Autoloading	13
4.3 Jmenné konvence	14
4.3.1 Názvy tříd	14
4.3.2 Adresářová struktura a názvy souborů	14
4.4 Architektura	15
4.4.1 Historie MVC	15
4.4.1.1 První návrh	15
4.4.1.2 Druhý návrh	15

4.4.1.3	Implementace v jazyku Smalltalk	15
4.4.2	MVC u webových aplikací	16
4.4.3	MVC v projektu	17
4.5	Umístění business logiky	18
4.5.1	Model	18
4.5.2	Action Controller	18
4.6	Object-relational mapping	18
4.7	Konfigurace	18
4.8	Scaffolding	19
4.9	Routing	19
4.10	Lokalizace	20
4.11	Logování	20
4.12	Ladění výkonu	20
4.13	Uživatelské typy	21
4.14	Komponenty	21
4.15	Šablonový systém	21
4.16	Zásadní třídy aplikace	22
4.16.1	Model	22
4.16.2	Object	22
4.16.3	View Helper	22
4.17	Bezpečnost	23
4.17.1	Cross-site scripting (XSS)	23
4.17.2	Cross-site request forgery (CSRF)	23
4.17.3	PHP injection	24
4.17.4	SQL injection	24
4.17.5	Session hijacking	24
4.17.6	Session fixation	25
4.17.7	Brute force attack	25
4.18	Doporučená konfigurace	25
4.18.1	Konfigurace PHP	25
4.18.1.1	Open_basedir	25
4.18.2	Konfigurace Apache	26
5	Realizace	27
5.1	Styl zápisu kódu	27
5.1.1	Formátování kódu	27
5.1.1.1	Odsazení a zarovnání kódu	27
5.1.1.2	Závorkování	28
5.1.2	Pojmenování	28
5.1.3	Dokumentace	28
5.1.4	Uvozovky	28
5.1.5	Vzorová ukázka	28
5.2	Pravidla pro tvorbu šablon	30
5.2.1	Vkládání PHP kódu	30
5.2.2	Povolené příkazy	30
5.2.3	Povolené funkce	30
5.3	Pravidla pro tvorbu PHP tříd	30
5.3.1	Kódování	30
5.3.2	Zamezení předčasného výstupu	31
6	Testování	33

6.1	Testování kódu	33
6.1.1	Tvorba testů	33
6.1.1.1	Princip	33
6.1.1.2	Formát testového souboru	33
6.2	Zhodnocení zvolených řešení	34
6.2.1	Nepoužití hotového frameworku	34
6.2.2	Srovnání s nejrozšířenějšími WCMS	35
6.2.3	Kompaktnost	35
6.2.4	Efektivita práce s aplikací	36
7	Závěr	37
7.1	Zhodnocení splnění cílů	37
7.2	Další pokračování projektu	37
8	Literatura	39
9	Přílohy	41
9.1	Seznam použitých zkratk	41
9.2	Use case diagramy	42
9.3	Návrhové diagramy a obrázky	45

Seznam obrázků

3.1	Doménový model	10
3.2	Stavy článku	11
4.1	První návrh MVC	15
4.2	MVC v jazyku Smalltalk	16
4.3	MVC v aplikaci	17
4.4	Dispatcher View	17
4.5	Konfigurační soubor	19
4.6	Zápis cest	20
4.7	Lokalizace	20
4.8	Uživatelské typy	21
4.9	Model	22
9.1	Celkový use case diagram	42
9.2	Správa obsahu	43
9.3	Správa uživatelů	44
9.4	Třídy pro práci s konfigurací	45
9.5	Průběh scaffoldingu	45
9.6	Scaffolding	46
9.7	Routing	46
9.8	Object	47

Seznam tabulek

6.1	Srovnání s objektovými WCMS	34
6.2	Srovnání s nejrozšířenějšími WCMS	35
6.3	Velikost archivu	35

1 Úvod

1.1 Zadání bakalářské práce

Cílem bakalářské práce je vyvinout kompaktní redakční systém určený pro dvě cílové skupiny: nenáročného uživatele a programátory. Nenáročným uživatelům bude systém umožňovat velice snadnou publikaci na internetu bez nutnosti znát webové technologie. Programátorům pak systém zajistí dobrý základ pro vývoj složitějších aplikací na míru. Důraz bude kladen především na vysoký výkon, snadnou rozšiřitelnost, jednoduchou instalaci, striktní dodržování objektového přístupu a softwarové architektury, přehledné uživatelské rozhraní a snadnou konfiguraci.

1.2 Úvod do problematiky

CMS neboli systém pro správu obsahu je software zajišťující správu dokumentů [36]. Pod označením CMS většinou myslíme webovou aplikaci, často pak označovanou jako WCMS¹. WCMS zahrnuje mnoho kategorií. V obecném pojetí se jakákoliv webová aplikace pracující s dokumenty dá považovat za WCMS. Za WCMS tedy můžeme považovat portály, e-mailové klienty, vyhledávače, webová úložiště (např. Flickr), podnikové informační systémy atd.

V této práci se zaměřím na úzkou skupinu aplikací, pro které se označení CMS vžilo nejvíce: redakční systémy. Redakční systém je nástroj umožňující vytvořit a následně spravovat webovou prezentaci. Ve většině případů umožňuje publikaci statických stránek, článků, fotogalerií, anket apod. Redakční systém je obvykle rozdělen na dvě části: uživatelskou a administrační.

Přestože není historie redakčních systémů nikterak dlouhá, zatěžuje současné nejrozšířenější aplikace silným břemenem. Kvůli zpětné kompatibilitě totiž není možné provést razantní změny v kódu, které by je přiblížily současným standardům návrhu a programování. Ani jeden významný redakční systém neobsahuje alespoň náznak jakékoliv osvědčené softwarové architektury (např. Model-View-Controller). Většina známých aplikací je psaná neobjektově a postrádá rozdělení kódu do více vrstev.

¹Web Content Management System

2 Popis problému, specifikace cíle

2.1 Cíl práce

Cílem bakalářské práce je provést a zdokumentovat vývojové práce na projektu NORS.

2.2 Cíl projektu NORS

Cílem projektu je vyvinout kompaktní redakční systém pro platformu PHP/MySQL. Cílem projektu není konkurovat komplexním CMS, ale spíše se soustředit na uživatele, jejichž požadavky žádný redakční systém zatím nesplňuje. Důraz bude kladen především na jednoduchost, vysoký výkon, snadnou rozšiřitelnost, jednoduchou instalaci, striktní dodržování objektového přístupu a softwarové architektury, přehledné uživatelské rozhraní a snadnou konfiguraci.

2.3 Důvody vzniku

Hlavním důvodem pro vznik projektu je neexistence kompaktního redakčního systému postaveného na jakékoliv osvědčené softwarové architektuře. Většina známých aplikací je psána neobjektově a postrádá rozdělení kódu do více vrstev.

2.4 Cílová skupina produktu

Systém bude určen pro dvě cílové skupiny: nenáročné uživatele a programátory. Nenáročným uživatelům bude systém umožňovat velice snadnou publikaci na internetu bez nutnosti znát webové technologie a číst dokumentaci. Programátorům pak systém zajistí velmi lehce a rychle pochopitelný základ pro vývoj složitějších aplikací na míru. Kód se bude řídit velice přísnými pravidly a měl by být srozumitelný i začínajícímu programátorovi.

2.5 Teze projektu

„Simplicity over complexity” - Celý projekt bude vyvíjen se snahou udržet maximální jednoduchost instalace, nastavení a obsluhy. Překombinovaných a nepřehledných projektů tohoto druhu existuje již mnoho.

„Plug and play” - Projekt bude ihned po stažení a jednoduché instalaci připraven použit. Nebude potřeba ruční konfigurace ani studium dokumentace.

Pro udržení kompaktnosti a jednoduchosti projektu bude použit princip „Convention over Configuration” [17], což je paradigma, které má za cíl použitím konvencí snížit velikost konfiguračních souborů a počet rozhodnutí, které musí programátor dělat. Konvencí je například pravidlo, že název tabulky v databázi má stejné jméno jako odpovídající Active Record třída. Dalším příkladem konvence je pevně daná adresářová struktura aplikace.

2.6 Rešerše existujících projektů

Redakčních systémů napsaných pro platformu PHP/MySQL existuje opravdu mnoho. Bohužel je většina systémů poznamenaná svou dlouhou historií nebo nezkušeností autora. Velkým překvapením pro mne byla špatná kvalita kódu projektu Joomla, který je pouhé tři roky starý a stojí za ním široká komunita.

Důležitým aspektem každého redakčního systému je jeho výkon. Proto byl každý z projektů podrobil základnímu výkonostnímu testu.

2.6.1 Metodika testu

Výkon, tedy vlastně procesorový čas potřebný k zobrazení stránky, byl měřen pomocí nástroje Jakarta jMeter [8]. Pro každý z WCMS bylo v jMeteru spuštěno 10 vláken. Každé vlákno následně provedlo 10 GET dotazů na úvodní stránku. U každého projektu je uveden průměrný čas v milisekundách potřebný k obslužení jednoho požadavku.

2.6.2 Srovnávané projekty

2.6.2.1 Drupal

Drupal [3] je dnes jedním z nejoblíbenějších CMS systémů vůbec. První verze byla napsána roku 2000 holandským studentem Driesem Buytaertem. V současné době se o další vývoj stará několik hlavních vývojářů a více než 400 dobrovolných přispěvatelů.

Klady

- instalační skript testuje funkčnost všech potřebných nastavení
- dobře zpracovaná a přehledná administrace

Zápory

- neumožňuje během instalace vytvořit novou databázi
- strukturální kód

Průměrný čas: 1486ms

2.6.2.2 Joomla!

Joomla! [16] je ze všech testovaných projektů nejmladší. První verze byla vydána v polovině roku 2005.

Klady

- kontroluje během instalace veškerá důležitá nastavení

Zápory

- rozsah zdrojových kódů (2MB)

- nepřehledná administrace
- velice složitý systém
- bez architektury
- promíchání všech tří vrstev
- pouze částečně objektový kód

Průměrný čas: 1403ms

2.6.2.3 Nucleus

Nucleus [18] je velice známý WCMS vyvíjený Wouterem Demuynckem od roku 2002.

Klady

- umožňuje administrovat více weblogů zároveň

Zápory

- instalace se nespouští automaticky
- administrace není příliš uživatelsky přívětivá
- částečně neobjektový kód
- zcela chybí architektura
- promíchání všech tří vrstev

Průměrný čas: 898ms

2.6.2.4 PHP-Nuke

PHP-Nuke [21] je dílo Francouze Francisco Burzi prvně vydané roku 2000.

Zápory

- rozsah zdrojových kódů (6MB)
- 96 tabulek v databázi
- instalace se nespouští automaticky
- administrace není příliš intuitivní
- strukturální kód

Průměrný čas: 5112ms

2.6.2.5 phpRS

phpRS [22] je český systém vyvíjený Jiřím Lukášem od roku 2001. Vzhledem k datu poslední verze (10.1.2007) se nejspíše jedná o mrtvý projekt.

Klady

- velice rychlý

Zápory

- instalace se nespouští automaticky (nutné přečíst dokumentaci)
- chybový instalační skript
- strukturální a velmi špatně čitelný kód

Průměrný čas: 85ms

2.6.2.6 RS2

RS2 [24] je opět český produkt, jehož autorem je Juneau.

Klady

- velmi přehledná administrace

Zápory

- neumožňuje během instalace vytvořit novou databázi
- strukturální kód

Průměrný čas: 1224ms

2.6.2.7 Typo3

Autorem tohoto stařečka je Kasper Skaarhoj. Vývoj systému zasahuje až do roku 1999. Typo3 [29] je řazen do kategorie Content Management Frameworků.

Zápory

- rozsah zdrojového kódu (7,5MB)
- velmi obtížná instalace
- objektový, ale bez architektury
- promíchání všech tří vrstev
- po instalaci není plně použitelný

Průměrný čas: 341ms

2.6.2.8 Wordpress

Kořeny Wordpressu [38] systému sahají do roku 2001. Jedná se o jeden z celosvětově nejznámějších a nejrozšířenějších CMS.

Klady

- přehledná administrace

Zápory

- neumožňuje během instalace vytvořit novou databázi
- strukturální kód

Průměrný čas: 3598ms

3 Požadavky a jejich analýza

3.1 Požadavky

Specifikace požadavků je jedním ze zásadních kroků při vývoji softwaru podle metodik unifikovaného procesu vývoje aplikací [15]. Důkladný sběr uživatelských požadavků a jejich správná analýza jsou rozhodující pro úspěch jakéhokoliv softwarového projektu.

Požadavky se rozdělují na funkční požadavky (požadavky na funkce softwaru) a na obecné požadavky.

3.1.1 Funkční požadavky

3.1.1.1 Back-end

- Administrátor bude moci vytvářet skupiny uživatelů.
- Administrátor bude moci měnit skupinám práva.
- Administrátor bude moci spravovat uživatelské účty a přiřazovat je ke skupinám.
- Administrátor bude moci měnit nastavení aplikace.
- Redaktor bude moci spravovat obsah (rubriky, stránky, články, komentáře).
- Redaktor bude moci ke článku připojovat obrázky.

3.1.1.2 Front-end

- Uživatel bude moci číst články, stránky, komentáře.
- Uživatel bude moci přidat komentář pod článek.
- Uživatel bude moci ohodnotit článek.

3.1.2 Obecné požadavky

- Pro client-side skriptování bude použit neinvazivní JavaScriptový framework jQuery.
- Kód aplikace bude rozdělen do více samostatných vrstev.
- Konfigurace a schémata tabulek budou uloženy jako YAML-like soubory.
- Konfigurace se bude dát zobrazit a editovat v podobě rozklikávacího stromu.

Funkční požadavky jsou zachyceny také v základním use case diagramu 9.1, který je následně detailněji rozkreslen v diagramech správy obsahu 9.2 a správy uživatelů 9.3. Diagramy jsou uvedeny v příloze.

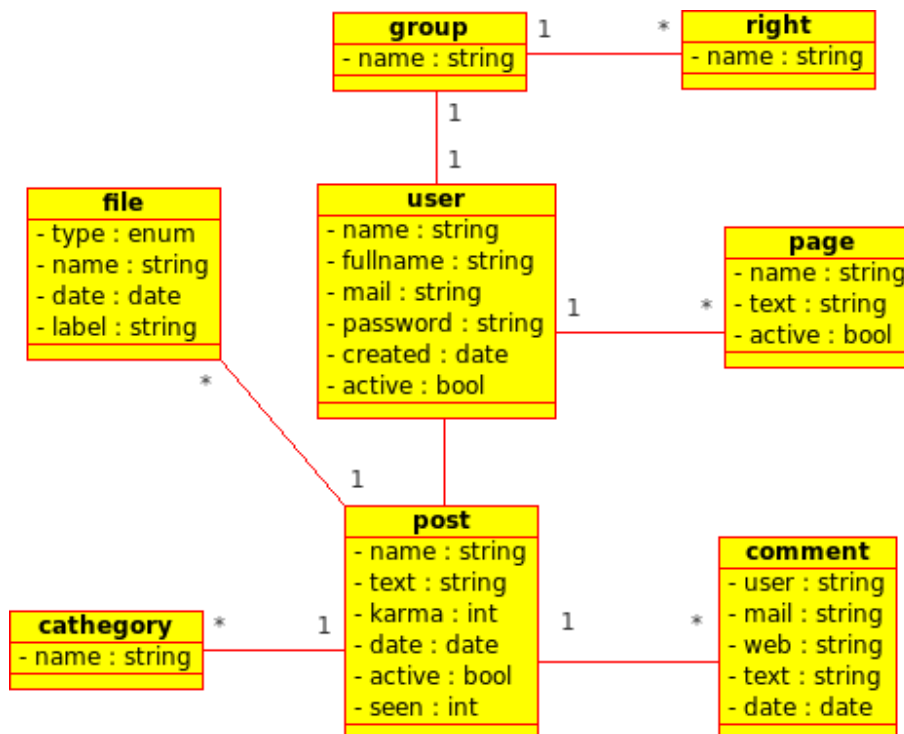
3.2 Analýza

Vzhledem k povaze celé práce, která se snaží soustředit především na návrh a implementaci, nebude analytická část rozebrána příliš detailně.

3.2.1 Doménový model

Doménový model je konceptuální model popisující entity systému a jejich vzájemné vztahy.

Základní doménou modelu je uživatel. Uživatel je návštěvník webové stránky, který má vytvořen uživatelský účet, a má proto přístup do administrace stránek. Každý uživatel patří do skupiny uživatelů. Skupina uživatelů má přiřazena určitá práva. Uživatel vlastní stránky a články, které vytvořil. Článek má přiřazeny komentáře a připojené soubory. Každý článek patří do nějaké kategorie.

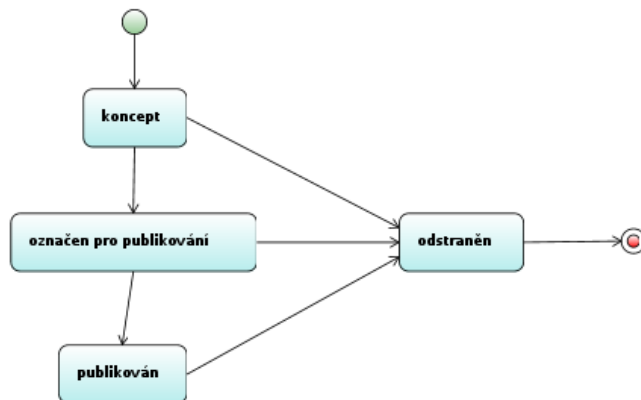


Obrázek 3.1: Doménový model

3.2.2 Stavové diagramy

Stavový diagram zachycuje chování systému pomocí popisu jeho stavů.

Článek se po vytvoření stává konceptem. Koncept může být uživatelem označen pro publikování. Ve chvíli, kdy článek dosáhne času publikování, je článek publikován. Článek může být v kterémkoliv ze stavů odstraněn.



Obrázek 3.2: Stavy článku

4 Návrh řešení

Těžištěm úspěchu tohoto projektu je právě precizní návrh, který by měl produktu zajistit silnou konkurenční výhodu.

4.1 Platforma

Jak již bylo dříve nastíněno, bude výsledkem projektu webová aplikace postavená nad platformou PHP5/MySQL. Aby bylo možné využít všech možností objektově orientovaného paradigmatu, bude vyžadován interpret PHP verze 5.2¹ nebo vyšší [10]. MySQL bude vyžadováno ve verzi alespoň 5. Z webových serverů bude podporován především Apache.

4.1.1 Nároky na konfiguraci

Aby mohla být v aplikaci použita dobře čitelná URL, bude vyžadována ze strany webserveru podpora .htaccess souborů a načtený modul Rewrite [7]. Pro umožnění práce s obrázky (vytváření náhledů) bude požadováno PHP rozšíření php_gd [11]. Více o doporučené konfiguraci bude uvedeno v části zabývající se bezpečností.

4.1.2 Framework

Jako základ webových aplikací se dnes silně doporučuje použít některý z osvědčených a rozšířených PHP frameworků. PHP framework je soubor knihoven, doporučených postupů a návrhových vzorů, který má za cíl usnadnit programátorům vývoj webových aplikací. Snaží se především řešit typické problémy webového prostředí jako je zpracování požadavku, stavovost, práce s databází, práce s formuláři, bezpečnost atd., aby se vývojář mohl soustředit pouze na zadání projektu.

Bohužel v době vytváření návrhu aplikace neexistoval žádný framework, který by vyhovoval nárokům a požadavkům aplikace. Frameworky byly buď příliš komplexní a měli vysokou režií, nebo neobsahovaly dostatečné odstínění od HTTP protokolu a nebo byly psány pro PHP 4. Proto bylo nutné napsat framework vlastní. Pokud by byla aplikace nyní navrhována znovu od základu, přicházel by v úvahu framework Nette [9], který mezitím velmi dospěl.

Inspirací pro základ nového frameworku se stal článek Understanding MVC in PHP [26], ze kterého bylo převzato především zpracování požadavku a autoloading.

4.2 Autoloading

PHP od verze 5 umožňuje definovat funkci `__autoload`, která je zavolána, když PHP interpret potřebuje načíst novou třídu a neví, kde ji hledat. Této funkci pak jako jediný argument předá právě název třídy. Díky této vlastnosti není potřeba na začátku všech skriptů psát seznam požadovaných souborů, stačí nadefinovat funkci `__autoload` a zvolit vhodné pojmenování tříd.

Aby byl autoloading co možná nejjednodušší, bude název všech tříd odpovídat jejich umístění. Pokud tedy bude třída ve složce Core/View v souboru Default.php, bude název třídy `Core_View_Default`. Takovýto název třídy pak půjde ve funkci `__autoload` velmi snadno převést na cestu k souboru a třídu následně načíst.

¹Nižší verze PHP měli velmi slabou podporu OOP.

4.3 Jmenné konvence

Aby mohl být v dalších částech práce rozebrán návrh až na úroveň jednotlivých tříd, je nejprve nutné nastínit pojmenování jednotlivých komponent. Jejich význam bude rozebrán později.

4.3.1 Názvy tříd

Všechny třídy frameworku budou umístěny v adresáři Core, a proto budou názvy tříd v těchto souborech podle pravidel autoloadingu obsahovat předponu Core_.

Pokud bude třída frameworku dědit od jiné třídy, bude název rodiče předřazen názvu třídy. Např. třída Form dědí ze třídy Helper, proto její celý název bude Core_Helper_Form.

Jelikož lesů ubývá a čtenářův čas je drahocenný, budou názvy tříd dále uváděny bez předpony Core_.

4.3.2 Adresářová struktura a názvy souborů

Jelikož většina hostingových uživatelů nemá možnost ukládat soubory níže než do DocumentRootu, budou všechny soubory aplikace umístěny do DocumentRootu a do adresářů v něm obsažených.

Názvy složek, které obsahují soubory s uživatelským obsahem (ActionControllery, šablony, css, js), budou začínat malým písmenem. Jména složek, které obsahují soubory frameworku, budou začínat velkým písmenem.

Názvy souborů, které obsahují PHP třídy, budou začínat velkým písmenem (CamelCase).

Všechny soubory obsahující PHP kód (i šablony) budou mít koncovku .php, aby nemohl být v prohlížeči klienta zobrazen jejich obsah. Často se totiž stává, že vkládané PHP soubory byly pojmenovány soubor.inc a zdrojový kód takovýchto souborů se pak dá snadno zobrazit. V případě, že v takovémto souboru jsou zapsány přístupové údaje k databázi, vzniká vážný problém.

Všechny soubory, které jsou součástí frameworku a uživatel aplikace by je neměl měnit, budou umístěny ve složce Core.

Ve složce **styles** budou podsložky pojmenované stejně jako názvy vzhledů (základní bude pojmenován "default"). To umožní uživateli velmi snadno zcela změnit vzhled aplikace.

- Controllery budou umístěny ve složce **controllers**.
- Modely budou umístěny ve složce **models**.
- Šablony budou umístěny ve složce **tpl**.
- Veškerá grafika (styly i obrázky) bude umístěna ve složce **styles**.
- Lokalizační soubory budou umístěny ve složce **locales**.

4.4 Architektura

Jako základ aplikace bude použita softwarová architektura Model-View-Controller [28]. Díky jejímu použití bude celá aplikace rozdělena na tři samostatné komponenty:

- Model - datový model aplikace
- View - pohled na Model realizovaný uživatelským rozhraním
- Controller - řídicí logika

Návrhový vzor MVC jako jeden z mála nemá úplně přesnou definici a byl implementován mnoha lehce odlišnými způsoby. Abychom správně pochopili veškeré souvislosti, je třeba shrnout si v krátkosti historii MVC.

4.4.1 Historie MVC

4.4.1.1 První návrh

Původní návrhy architektury MVC [23] z roku 1979 se dnes téměř nepoužívá. Sám autor návrhů Trygve Reenskaug si uvědomil, že problematika MVC je v současnosti daleko širší než v roce 1979, a snaží se vytvořit nový návrh [13].

V prvotním dokumentu z 12.května 1979 je architektura pojmenována Model-View-Editor, kdy Editor je rohraní mezi uživatelem a jedním nebo více Views. Editor je v tomto případě tedy to, co dnes nazýváme nejčastěji Controller.



Obrázek 4.1: První návrh MVC

4.4.1.2 Druhý návrh

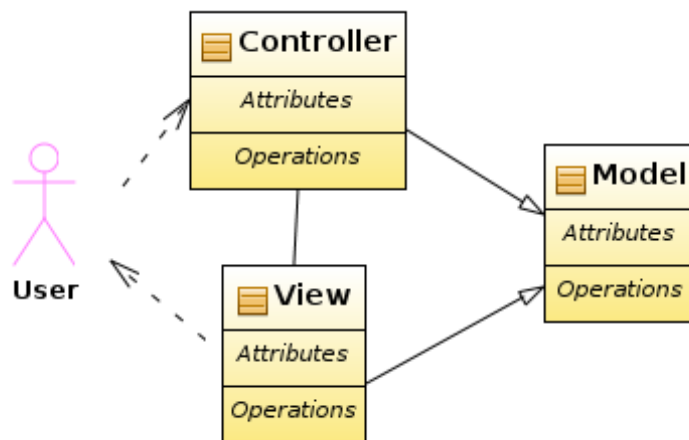
Druhý návrh z 10.prosince téhož roku již nese známý název Models-Views-Controllers, ale ve skutečnosti sestává ze čtyř částí: Models, Views, Controllers a Editors. Editor je součástí View, která se chová jako rozšíření Controlleru, a umožňuje uživateli změnit informace prezentované pomocí View.

4.4.1.3 Implementace v jazyku Smalltalk

První implementace MVC byla realizována roku 1987 firmou Xerox PARC v jazyku Smalltalk [1] a právě ona je dnes většinou brána jako referenční. MVC mělo za jistěovat rozdělení uživatelského vstupu (Controller), modelování vnějšího světa (Model) a vizuální prezentaci (View) na tři samostatné části.

Jsou zde zavedeny pojmy pasivní a aktivní Model. Pasivní model se sám nijak aktivně nezapojuje do komunikace s ostatními komponentami, pouze reaguje na jejich požadavky. Aktivní model je potřeba v případě, že se Model může měnit, aniž by o to byl požádán View nebo Controllerem. V případě změny Model upozorní View (a další registrované závislosti) na změnu svého stavu.

Model je jako členská proměnná uložen nejen ve View, ale i v Controlleru, který tak může měnit jeho stav. Model neobsahuje žádný přímý odkaz na View nebo Controller, umí pouze poslat zprávu o změně stavu svým registrovaným závislostem.



Obrázek 4.2: MVC v jazyku Smalltalk

4.4.2 MVC u webových aplikací

Původní návrhy MVC i první implementace byly určeny pro desktopové prostředí². Proto můžeme původní návrhy u desktopových aplikací beze změn nadále používat. U webových aplikací je ale situace odlišná. Webové aplikace neběží v nekonečné smyčce čekající na aktivitu uživatele ale na straně serveru v klient-server komunikaci. Z toho vyplývá, že aplikace musí být schopna vytvořit požadovaný výstup (reflektující předchozí stav) pouze na základě URL a HTTP hlaviček. To samozřejmě přináší zvýšené nároky především na Controller ale také na rychlost sestavení celé aplikace.

Vzhledem k tomu, že po odeslání výstupu klientovi se ukončí HTTP spojení a celá aplikace se ukončí, není spojení Modelu na View snadno realizovatelné a ve výsledku není ani potřeba. Pokud bychom přesto potřebovali, aby mohl Model odesílat View aktualizace stavu, museli bychom ponechat trvale otevřené HTTP spojení, protože navázání komunikace není ze strany serveru možné kvůli skrytí klientských počítačů za překladače adres a firewally. Ponecháním otevřených spojení (a běžících aplikací) ale dochází ke zvýšené spotřebě systémových prostředků, čemuž se snažíme na serveru vyhnout.

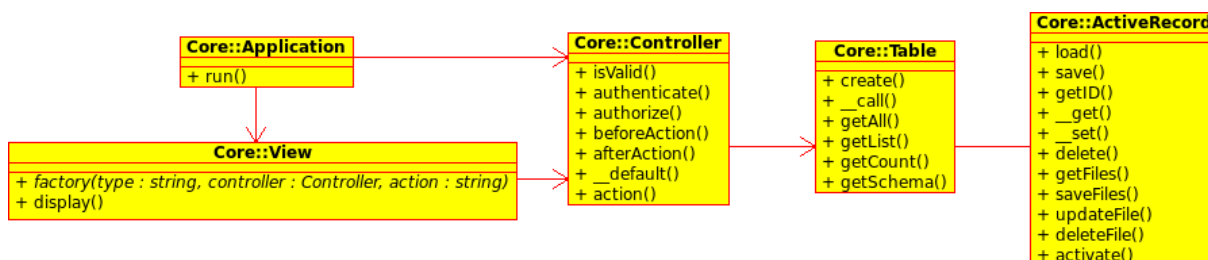
Kvůli potřebě předzpracování požadavku a nutnosti znovuvybudování celé aplikace při každém požadavku není možná ani přímá vazba z View na Model. Veškeré požadavky od uživatele musejí procházet "skrz" Controller. Pokud potřebujeme v aplikaci aktualizovat část View aniž by o to uživatel musel požádat znovunačtením stránky, pomůže nám technologie AJAX, která umožní vytvořit na pozadí bez zásahu uživatele nový HTTP požadavek s cílem načíst pouze změněná data.

²ostatně Internet v roce 1979 ještě vůbec neexistoval

4.4.3 MVC v projektu

Architektura aplikace vychází z článku Understanding MVC in PHP [26] a velmi se podobá architektuře frameworků Zend Framework a CakePHP.

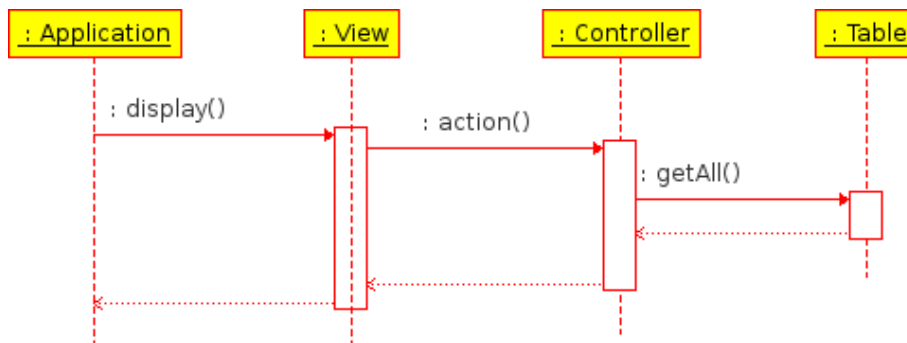
Jak bylo zmíněno dříve, hraje u webových aplikací velmi důležitou roli Controller. V projektu bude Controller realizován pomocí Front Controller návrhového vzoru [27], což ve stručnosti znamená, že veškerá komunikace ze strany uživatele bude procházet přes jediný vstupní PHP skript. Front Controller bude sestávat z bootstrap³ skriptu index.php a třídy `Core_Application`. Podle volaného URL a HTTP hlaviček `Core_Application` vytvoří odpovídající Action Controller a View.



Obrázek 4.3: MVC v aplikaci

Action Controller je základem zpracování každého požadavku. Jeho veřejným metodám se říká Actions. Název Action Controlleru a jeho metod odpovídá URL požadavků, které má Action Controller obsloužit. Např. URL `domain.tld/user/add/` obslouží metoda `add` třídy `User`, URL `domain.tld/user/list/` obslouží metoda `list`.

Ve většině webových frameworků předává Front Controller řízení Action Controlleru, zde ale bude řízení předáno metodě `display` třídy `View`. Tato metoda následně rozhodne, jestli bude uživateli odeslána uložená cache a nebo jestli se řízení předá Action Controlleru. V případě, že se odešle cache, nedojde vůbec k zavolání metody Action Controlleru a tedy ani k volání Modelu, čímž se velmi výrazně sníží čas potřebný k vyřízení požadavku a spotřebované zdroje. Tento způsob předání řízení je známý pod názvem Dispatcher View [19].



Obrázek 4.4: Dispatcher View

Model bude založen na Active Record [20] návrhovém vzoru, který zajistí správné zapouzdření Modelu a mapování relačního schématu na objekty.

³Zaváděcí skript

4.5 Umístění business logiky

Business logika je definována jako soubor algoritmů použitých pro výměnu informací mezi databází a uživatelským rozhraním [31].

Business logika aplikace bude rozdělena do dvou vrstev:

- Action Controller (logická vrstva)
- Model (datová vrstva)

4.5.1 Model

Model bude obsahovat logiku, která úzce souvisí s datovým modelem. Veškeré SQL dotazy se budou nacházet právě v Modelu, což zajistí zachování integrity dat i při použití jiného Controlleru a View (např. terminálu).

4.5.2 Action Controller

V Action Controlleru bude logika, která bude sloužit k předzpracování dat pro Model a View.

V případě předzpracování pro Model se bude jednat například o naplnění Active Record objektu daty z POST.

Předzpracování pro View bude například vytváření náhledů pro obrázky u článku nebo sestavování URL odkazů na jednotlivé stránky.

4.6 Object-relational mapping

Jako rozhraní mezi objektovou aplikací a relační databází bude použit návrhový vzor Active Record [30]. Ten umožní pracovat s jednotlivými záznamy databáze, jako by to byly objekty. Jednotlivé sloupce tabulek se stanou členskými proměnnými Active Record objektů.

Pro pohodlnou práci s SQL dotazy existují Object-relational mapping frameworky (např. Doctrine). Ty umožňují sestavovat dotaz pomocí volání metod.

```
$query->from('users');  
$query->orderby('username');  
$query->execute();
```

V aplikaci žádný takovýto framework použit nebude. Důvodem je znatelné snížení výkonu a nárůst velikosti archivu. Mimoto jazyk SQL je sám o sobě velmi komfortní nástroj pro dotazování, není potřeba vkládat mezi něj a aplikaci ještě další vrstvu.

4.7 Konfigurace

Pro maximálně snadnou a uživatelsky příjemnou konfiguraci bude vytvořen jeden konfigurační soubor se syntaxí podobnou jazyku YAML. Konfigurace bude sestávat z klíč:hodnota záznamů, kdy hodnota bude moci být dvou typů: řetězec nebo slovník. Slovník je v tomto kontextu vnořená množina záznamů, kdy vnoření se vyjadřuje odsazením.

Takto vytvořená konfigurace lze velmi snadno převést na PHP pole a to uložit do souboru, aby se konfigurace nemusela parsovat při každém spuštění. Konfiguraci bude možno měnit i přímo v administraci.



Obrázek 4.5: Konfigurační soubor

Konfigurace bude přístupná skrze třídu Config a bude měnitelná za běhu. Třídy podílející se na práci s konfigurací jsou zachyceny na diagramu 9.4.

4.8 Scaffolding

Jelikož se předpokládá, že si uživatelé-programátoři budou chtít dodělat do aplikace další funkcionalitu, bude pro vytváření tříd Modelu využit princip Scaffoldingu [35]. Uživatel pouze vytvoří schéma tabulky (YAML-like soubor ve složce models/schemas) a aplikace už si sama vygeneruje potřebné ActiveRecord a Table třídy a vytvoří v databázi tabulku. Navíc uživatel nebude muset explicitně žádat o generování tříd, aplikace si je vytvoří sama v okamžiku, kdy je bude potřebovat. Průběh scaffoldingu je zachycen v diagramu 9.5.

Pomocí této funkcionality lze také například upravit tabulku v databázi tím, že pouze změníme schéma tabulky a smažeme tabulku v databázi. Jakmile bude aplikace tabulku potřebovat, sama ji podle schématu vytvoří. Editační formuláře v administraci se navíc také sestavují s pomocí schémat, takže nemusíme nic upravovat.

Podoba schématu tabulky a ukázka vygenerované třídy jsou zachyceny na obrázku 9.6.

4.9 Routing

Jedním z důvodů, proč nebyl použit framework CakePHP, bylo nedostatečné odstínění od HTTP protokolu. Programátor by při vytváření odkazů musel neustále myslet na formát URL a nemohl by URL snadno najednou změnit v celé aplikaci.

V aplikaci bude pro vyřešení tohoto problému použito routování URL. V aplikaci se nebude nikde pracovat přímo s URL, ale pouze s vnitřním požadavkem aplikace. Vnitřní požadavek sestává z názvu Action Controlleru, události a případných dalších parametrů. O vytvoření URL z vnitřního požadavku se postará třída Router. Stejně tak bude Router schopen vytvořit vnitřní

požadavek z URL. Jedná se tedy o dvoucestný Router. Formát jednotlivých cest (route) bude zapsán v konfiguračním souboru.

```

routes:
  → default:
    → → format: $action/$event
    → → defaults:
      → → → action: post
      → → → event: __default

```

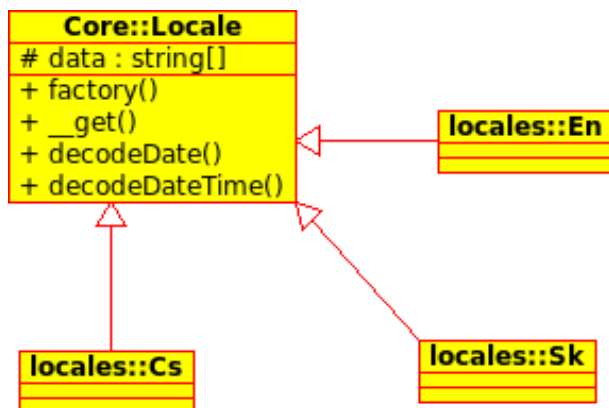
Obrázek 4.6: Zápis cest

```
$router->genURL('post', 'view', $id); --> http://domain.tld/post/1/
```

Vazby mezi controllery a třídami zajišťujícími routing jsou zachyceny v diagramu routování 9.7.

4.10 Lokalizace

Aplikace bude umožňovat snadnou lokalizaci a přepínání mezi jazykovými verzemi. Pro tento účel bude pro každý jazyk vytvořen jeden lokalizační soubor, který bude obsahovat překlady textů aplikace a funkce pro lokalizaci času a datumu. V celé aplikaci se pak bude vypisování textů obalovat do funkce `__()`, která zajistí překlad do správného jazyka.



Obrázek 4.7: Lokalizace

4.11 Logování

Veškeré chybové zprávy se budou zapisovat do souboru nastaveného v konfiguraci. Logování bude zajišťovat třída Log, kterou bude využívat funkce zpracovávající výjimky.

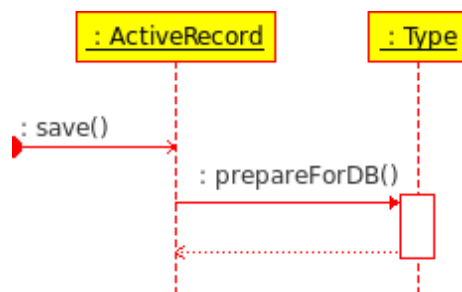
4.12 Ladění výkonu

Pro lepší přehled o využitých prostředcích bude aplikace umožňovat zobrazit množství využité paměti, dobu zpracování požadavku, počet načtených souborů, počet dotazů do databáze a

využití cache. V případě potřeby vypíše také seznam všech provedených SQL dotazů s jejich dobami běhu.

4.13 Uživatelské typy

Zejména pro potřeby validace vstupních dat bude použita technika uživatelských typů. Pro každý datový typ, který uživatel zadá do schématu tabulky, musí existovat odpovídající třída, která zajistí vyčištění hodnoty před vložením do SQL dotazu.



Obrázek 4.8: Uživatelské typy

Díky tomu se budeme moci spolehnout, že když ve schématu nadefinujeme sloupec jako datetime, opravdu se nám do databáze bude ukládat datetime a to například včetně převodu z lokalizované podoby (21.1.2009) do podoby vyžadované databází (2009-1-21).

Třídy uživatelských typů se budou využívat také při automatickém vytváření tabulek a vyplňování výchozích hodnot do polí formulářů.

4.14 Komponenty

Již bylo řečeno, že aplikace bude umožňovat zobrazení HTML cache místo předání řízení do Action Controlleru. Cache by se ale nedala použít v případě, kdy by na stránce byly uživatelsky závislé údaje, např. jméno přihlášeného uživatele. Proto budou moci být ve View použity Komponenty, které se budou vykreslovat dynamicky i přesto, že bude zbytek stránky načítán z cache.

Komponentu tedy můžeme definovat jako samostatnou nezávislou část webové stránky, jakýsi HTML fragment. Další vlastností Komponent bude, že se jejich public metody budou dát, stejně jako metody Action Controlleru, zavolat z webové stránky. Metody Komponent budou určeny k volání pomocí technologie AJAX.

Díky této technice budeme moci mít na úvodní webové stránce, která se kvůli vytížení bude cachovat, přihlašovací formulář. Odesláním formuláře se odešle AJAX požadavek na metodu Komponenty, která jako odpověď odešle HTML kód, který se vloží místo formuláře. Tím ušetříme nejen výkon (cachování) ale i konektivitu (měníme jen část stránky).

4.15 Šablonový systém

Šablonový systém je vrstva mezi aplikační a prezentační logikou aplikace, která zajišťuje jejich důkladné oddělení. Nejznámější šablonový systém v jazyku PHP je Smarty.

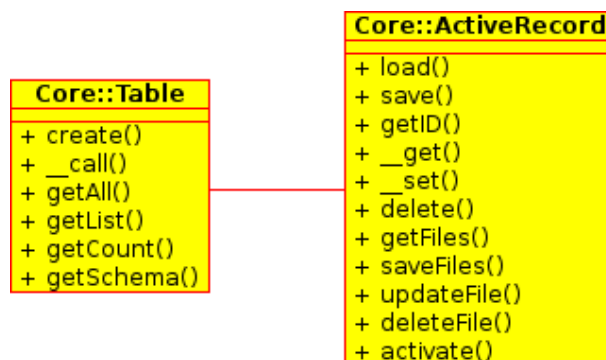
Aby bylo možné zachovat aplikaci dostatečně kompaktní a nenáročnou na zdroje, bude nutné jako šablonový systém/jazyk použít samotné PHP. To ale samozřejmě bude mít za následek zvýšené nároky na disciplínu kodérů, protože budou mít přístup k mnohem širší funkcionalitě než potřebují. Pravidla pro tvorbu šablon budou uvedeny později.

Šablonový systém bude mít na starost vyčištění dat z Action Controlleru a jejich předání do šablony.

4.16 Zásadní třídy aplikace

4.16.1 Model

Model bude sestávat ze dvou druhů tříd: Active Record a Table.



Obrázek 4.9: Model

Active Record třídy budou sloužit k práci s jednotlivými entitami (záznamy relační databáze). Budou umožňovat vytvářet nové entity, měnit a mazat již existující entity, nahrávat soubory atd.

Table třídy pak budou zajišťovat provádění SELECT dotazů nad tabulkami a budou vracet pole Active Record objektů.

Table třídy budou obsahovat magickou metodu `__call`, která se volá v případě, že není nalezena požadovaná metoda. Díky této vlastnosti bude možno volat například metodu `findBy-Name($value)`, kde za `Name` můžeme dosadit libovolný název sloupce tabulky a metoda nám vrátí pouze záznamy, kde hodnota sloupce odpovídá `$value`.

4.16.2 Object

Object bude abstraktní třída, ze které budou dědit všechny Action Controllery. Bude umožňovat svým potomkům přístup k základním objektům aplikace (Request, Response, Router, Config) a umožní jim používat reflexi. Vztahy s ostatními třídami jsou zachyceny v diagramu 9.8.

4.16.3 View Helper

Pro usnadnění vytváření prvků View, budou použity View Helpery. View Helpery umožní jednodušší práci s formuláři, navigací, stránkováním nebo jakýchkoliv jiných HTML prvků.

4.17 Bezpečnost

Bezpečnost webových aplikací je velmi ožehavé a rozsáhlé téma. Zejména aplikacím, které umožňují vkládání dat anonymními uživateli (např. komentáře), hrozí velké nebezpečí útoku. Typů zranitelností je dnes známo velmi mnoho a stále se objevují nové. V této práci se zaměříme pouze na několik nejdůležitějších zranitelností a na způsoby jejich řešení.

4.17.1 Cross-site scripting (XSS)

Jedná se o techniku, kdy útočník využije chybu ve skriptech aplikace k zobrazení vlastního Javascriptového nebo HTML kódu na stránce. Tento kód v lepším případě pouze poškodí vzhled nebo funkcionalitu stránky, v horším případě slouží k získávání citlivých údajů o uživateli (cookie) a překonávání bezpečnostních prvků aplikace. XSS zneužívá důvěru uživatele k určité webové stránce [33].

Na úrovni frameworku je XSS zamezeno vyčištěním výstupu pomocí funkce `Htmlspecialchars`. Toto čištění se provádí automaticky v Action Controlleru při volání metody `setData`, tedy během přípravy dat pro šablonu. Pokud víme, že data XSS neobsahují, a potřebujeme zachovat HTML značky v datech obsažené, můžeme to explicitně metodě `setData` nařídit pomocí třetího parametru (`allowHtml`).

Na úrovni aplikace je XSS zabráněno odstraněním HTML kódu z komentářů. Povoleno je pouze několik speciálních značek (`[b]`, `[i]`, `[code]`, `[img]` a `[url]`), které jsou při výpisování nahrazeny odpovídajícími HTML tagy.

4.17.2 Cross-site request forgery (CSRF)

CSRF je druh útoku, při kterém prohlížeč uživatele provede bez jeho vědomí akce na určité stránce. Obecně CSRF zneužívá důvěru stránky k prohlížeči uživatele [32].

Mějme příklad: Uživatel se přihlásí do internetového bankovníctví. Nechá záložku otevřenou a v druhé záložce si otevře fórum nachylné na útok typu XSS. Na toto fórum předtím útočník umístil příspěvek, který obsahuje obrázek. Zdroj obrázku ale nesměřuje na opravdový obrázek, ale na stránku uživatelské banky, kde se mění heslo (`banka.cz?newpass=xxx`). Jelikož je uživatel stále přihlášen (platná cookie) a požadavek na změnu hesla je odeslán z jeho prohlížeče, nepozná bankovní aplikaci, že požadavek neodeslal sám uživatel, a heslo změní.

CSRF lze samozřejmě provést i jinak než vložením obrázku. Můžeme například pomocí Javascriptu vyvolat asynchronní HTTP požadavek (AJAX) metodou POST, a tak simulovat odeslání formuláře uživatelem. Jediné, co nám k tomu stačí, je opět jakýkoliv web náchylný na XSS, na který uživatel chodí.

Protože nejsme schopni zajistit, aby byly odstraněny náchylnosti k XSS na všech stránkách Internetu, musí se proti CSRF bránit samotná cílová aplikace. Jednou z metod, jak se proti CSRF bránit je kontrola HTTP hlavičky `REFERRER` (URL stránky, z které uživatel přichází). Odesílání této hlavičky může ale uživatel vypnout, a proto tato metoda není příliš použitelná. Pravděpodobně jedinou spolehlivou metodou je do každého odkazu a formuláře, který způsobuje změnu dat, přidávat jedinečný token, který je následně při zpracování požadavku zkontrolován.

Přestože aplikace nemá na první pohled stejnou důležitost jako např. internetové bankovníctví, je její důkladné zabezpečení proti CSRF útokům velmi důležité. Smazání obsahu webových stránek nebo změna hesla uživatele může totiž mít pro uživatele velmi zásadní význam. Existují totiž například blogeri, pro které představuje psaní článků hlavní finanční zdroj.

V aplikaci je ochrana proti CSRF zabudována na úrovni frameworku v podobě nepovinného parametru metod na tvorbu URL (Router::genURL, redirect, forward). Pokud je parametr zadán, je přidán do URL token (hash) vzniklý kombinací náhodného čísla, session identifikátoru a otisku hesla uživatele. Na konec hashe je přidáno použité náhodné číslo. Při zpracování požadavku musí být zavolána metoda Request::checkCSRF, která zkontroluje správnost GET parametru.

Na úrovni aplikace jsou pak ještě zabezpečeny formuláře v administraci, do kterých je opět doplněn token.

4.17.3 PHP injection

PHP injection je útok na webovou stránku, která do skriptu(index.php) vkládá další PHP skripty, jejichž jméno získá z GET. Stránka index.php?page=home.php tedy vloží skript home.php. Pokud je to v konfiguraci PHP povoleno, můžeme tímto způsobem načíst i PHP skript z jiného serveru.

```
index.php?page=http://1.1.1.1/skript.php
```

Takto podvržený skript pak může například vypsat zdrojový kód skriptů uživatele nebo vypsat konfigurační soubor s heslem k databázi.

Ochrana v aplikaci spočívá v tom, že veškeré vkládané soubory jsou načítány s absolutní cestou k souboru.

4.17.4 SQL injection

SQL injection je útok, při kterém útočník pozmění vstupy takovým způsobem, že dojde k narušení správného zpracování SQL dotazu.

Například při změně svých údajů na stránce může útočník pozměnit vstup tak, aby se stal administrátorem.

```
"UPDATE users SET name = '". $_POST['name'] ."' WHERE ..."
```

Pokud útočník zadá jméno:

```
injector', admin = '1
```

stane se administrátorem.

Pro zamezení SQL injection jsou vstupy od uživatele ošetřeny přidáním zpětných lomítek před uvozovky a apostrofy.

4.17.5 Session hijacking

Session hijacking je útok, při kterém útočník získá session identifikátor uživatele a tím i přístup do chráněné oblasti webových stránek (administrace). Session identifikátor získá z PHPSESSID cookie pomocí Javascriptového kódu, který zobrazí na webových stránkách. Aby byl útok realizovatelný, musí být web napadnutelný pomocí XSS.

Tomuto útoku můžeme zabránit už při získávání session identifikátoru, a to tím, že zakážeme přístup k PHPSESSID cookie z Javascriptu.

`session.cookie_httponly`

Útoku můžeme zabránit i při autentizaci uživatele tím, že budeme kromě session identifikátoru kontrolovat i IP adresu uživatele. To je ale bohužel nepoužitelné pro některé uživatele mobilních připojení, protože těm se může IP adresa měnit velmi často.

4.17.6 Session fixation

Session fixation je méně známý útok, který spočívá v podvržení PHPSESSID cookie. Útočník nejprve musí zneužít XSS zranitelnosti stránek a pomocí JavaScriptu vytvořit uživateli novou cookie, která se jmenuje stejně jako session id cookie. Hodnotu cookie si útočník sám zvolí. Jakmile se uživatel přihlásí do chráněné části webových stránek, může útočník použít podvržené session id a ukrást uživateli session. Tím získá útočník přístup do chráněné části stránek.

Základní ochrana spočívá stejně jako u techniky Session hijacking v kontrole IP adresy. Lepším řešením je ale při změně stavu uživatele (přihlášení, odhlášení) vytvořit nové session id. Toho dosáhneme voláním funkce `session_regenerate_id`.

4.17.7 Brute force attack

Útok hrubou silou je technika pro zjištění hesla uživatele. U webových aplikací jde především o roboty, kteří se opakovaně pokoušejí projít přes přihlašovací formulář stránek. Tento útok je nebezpečný především v případě, kdy nekontrolujeme kvalitu hesel uživatelů.

Nejlepší metodou ochrany je zablokovat uživateli po N neúspěšných pokusech o přihlášení dočasně přihlašovací stránku. Ochrana proti tomuto druhu útoků bude implementována až v dalších verzích aplikace.

4.18 Doporučená konfigurace

Na začátku této kapitoly byla nastíněna minimální konfigurace stroje, na kterém má být aplikace provozována. Zde se zaměříme na konfiguraci doporučenou, tedy na to, jak učinit naši aplikaci bezpečnější a výkonnější.

4.18.1 Konfigurace PHP

PHP by mělo být nakonfigurováno s vypnutými direktivami `register_globals` a `safe_mod`, jak je ostatně doporučováno přímo vývojáři PHP [12].

4.18.1.1 Open_basedir

`Open_basedir` je direktiva PHP, která zabraňuje čtení souborů mimo nastavený adresářový strom. Důležitá je zejména na sdílených serverech (webhosting). Aby mohla aplikace měnit konfigurační soubor, musí mít práva čtení a zápisu do souboru. Bohužel na většině hostingů jsou skripty uživatelů spouštěny pod stejným uživatelem, takže pokud není správně nastavena direktiva `open_basedir`, budou moci ostatní uživatelé hostingu přepsat konfigurační soubor.

4.18.2 Konfigurace Apache

Pro zvýšení bezpečnosti je doporučováno provozovat PHP jako FastCGI aplikaci a používat SuExec pro spouštění skriptů pod právy uživatele.

Jako nebezpečná je označována HTTP metoda TRACE, a proto by měla být vypnuta.

5 Realizace

5.1 Styl zápisu kódu

Po oficiálním vydání stabilní verze projektu se předpokládá zájem ze strany uživatelů o doplňování projektu o jejich vlastní vylepšení. Proto, aby byla jednotnost zápisu kódu zachována i nadále, je nutné definovat závazná pravidla pro vývojáře.

5.1.1 Formátování kódu

Způsob odsazení a závorkování kódu se shoduje se stylem BSD KNF [34].

5.1.1.1 Odsazení a zarovnání kódu¹

- Délka řádku by neměla přesahovat 80 znaků.
- Aby nebyla omezena svoboda uživatelů v nastavení hloubky odsazení, je veškerý kód odsazen pravými tabelátory (hard tab). Šířka tabelátoru není nijak omezena.
- Pokud má funkce mnoho parametrů (řádek překročí 80 znaků), zarovnají se parametry s použitím mezer pod sebe. Stejně tak se zarovnají pod sebe části složité podmínky, prvky pole, části dlouhého řetězce apod.

```
function.foo($bar,
.....$bar2);
```

- Logická struktura kódu je posilována zarovnáním společných částí (=, ||) řádků pod sebe.

```
$foo...=.5;
$babar...=.6;
```

- U výrazů řídicích struktur je před a za podmínku přidána jedna mezera.

```
if.($foo==.1){
```

- SQL dotazy se zpřehledňují použitím zarovnání.

```
$query.="SELECT. 'name' ,
.....'pass'
.....FROM. 'users'
.....WHERE. 'id_user' =. '1'"
```

- Před a za operátory je přidána jedna mezera (platí i pro operátor "tečka").

```
$where = "WHERE 'id' = ' ' . $id . "'";
```

¹V ukázkách je znak mezera vyjádřen tečkou, znak tabelátor dvěma většitky ».

5.1.1.2 Závorkování

- U deklarací tříd, funkcí a metod je složená otevírací závorka na novém řádku.

```
class.Foo
{
}
```

- U výrazů řídicích struktur (podmínky, cykly) je složená otevírací závorka na stejném řádku.

```
foreach($arr.as.$item){
```

5.1.2 Pojmenování

- Veškeré názvy a komentáře jsou anglicky a neobsahují diakritiku.
- Jména proměnných, tříd a funkcí/metod by měly být smysluplné a co možná nejkonkrétnější.
- Názvy proměnných a funkcí/metod začínají malým písmenem a používají camelCase.
- Názvy tříd začínají velkým písmenem a také používají CamelCase. Pokud třída dědí od jiné třídy, měl by být název rodiče předřazen před název potomka, přičemž mezi oba názvy vložíme podtržítka (NázevRodiče_NázevPotomka).
- Klíčová slova SQL a názvy PHP konstant jsou psány velkými písmeny (včetně TRUE a FALSE).

5.1.3 Dokumentace

Všechny třídy, metody, funkce atd. budou dokumentovány pomocí PHPDoc [4].

5.1.4 Uvozovky

- Pokud PHP řetězec neobsahuje substituce je ohraničen jednoduchými uvozovkami.
- SQL dotazy jsou ohraničeny dvojitými uvozovkami.
- Názvy tabulek a sloupců v SQL dotazech jsou obaleny zpětnými uvozovkami ('users').

5.1.5 Vzorová ukázka

```
<?php
/**
 * ActiveRecord_User
 * @author Daniel Milde <daniel@milde.cz>
 * @copyright Daniel Milde <daniel@milde.cz>
 * @license http://www.opensource.org/licenses/gpl-license.php
 * @package Core
 */
```

```
/**
 * ActiveRecord_User
 * @author Daniel Milde <daniel@milde.cz>
 * @package Core
 */
class ActiveRecord_User extends Core_ActiveRecord
{
    /**
     * Array of fields in table
     * @var string[] $fields
     */
    public $fields = array('id_user' => array('visibility' => 0, 'type' => 'id'),
                          'name' => array('visibility' => 1, 'type' => 'text'),
                          'password' => array('visibility' => 1, 'type' => 'password'));

    /**
     * Constructor
     * @param int $id_user ID of the user
     * @return Core_ActiveRecord_User
     */
    public function __construct($id_user = FALSE)
    {
        parent::__construct('user', $id_user);
    }

    /**
     * Saves the object to DB
     * @return Core_ActiveRecord_User
     */
    public function save()
    {
        if ($this->id_user != 0) {
            $this->update();
        } else {
            $this->insert();
        }
        return $this;
    }
    ...
}
```

5.2 Pravidla pro tvorbu šablon

Jelikož bylo jako šablonový jazyk použito PHP, je nutné definovat pravidla pro tvorbu šablon, která zajistí oddělení prezenční a business vrstvy a zachování architektury MVC.

5.2.1 Vkládání PHP kódu

PHP kód bude vždy uvozen dlouhým otevíracím tagem (`<?php`).

Zkrácený tag (`<?`) není doporučován, protože neumožňuje použití XML a XHTML. ASP-style tag (`<%`) není doporučován, protože nemusí být na hostingu podporován.

PHP kód bude vždy ukončen uzavíracím tagem (`?>`)

5.2.2 Povolené příkazy

V PHP kódu bude povoleno použití pouze několika základních příkazů:

- `echo`
- `if`
- `foreach`
- `switch`

5.2.3 Povolené funkce

V PHP kódu bude povoleno použití těchto funkcí:

- `isset($var)` - vrátí TRUE pokud je proměnná (index pole) inicializovaná, jinak FALSE
- `iterable($array)` - vrátí TRUE, pokud je proměnná pole a obsahuje alespoň jeden prvek, jinak FALSE
- `__($str)` - přeložení řetězec do správného jazyka
- metody View helperů

5.3 Pravidla pro tvorbu PHP tříd

Aplikace je navržena tak, aby měl každý uživatel možnost ji snadno rozšiřovat a upravovat. Všichni uživatelé by měli při psaní nových tříd dodržovat několik pravidel.

5.3.1 Kódování

Pro zachování přenositelnosti budou veškeré soubory uloženy s kódováním UTF8. Použití multi-bytového kódování sice způsobí mírný nárůst velikosti souborů, ale na druhou stranu zaručí, že projekt budou moci používat všichni obyvatelé této i jiné planety (včetně Klingonů).

5.3.2 Zamezení předčasného výstupu

Aby nedocházelo k předčasnému odeslání výstupu (a HTTP hlaviček), je potřeba, aby každý soubor třídy obsahoval na samém začátku otevírací tag PHP (`<?php`). Ze stejného důvodu nesmí být použit Unicode BOM, který se používá pro snazší rozpoznání kódování souboru.

Jelikož některé editory vkládají na konec souboru prázdný řádek, nebude se na konec souborů s PHP třídami vkládat uzavírací znak (`?>`). Znak nového řádku mimo PHP blok by totiž také vedl k předčasnému odeslání výstupu.

6 Testování

6.1 Testování kódu

Aby bylo možné odhalit co možná nejvyšší procento chyb, bude kód zpětně pokrýván unit a integračními testy.

Unit test je nástroj sloužící k otestování správné funkčnosti dílčí části kódu (třídy, funkce, metody).

Integrační testy se zaměřují na testování korektní spolupráce dílčí části kódu s okolními součástmi.

Unit a integrační testy se nebudou používat pouze při vývoji nového kódu, ale také při úpravách stávajícího. Testování funkčnosti kódu po úpravách bývá označováno jako regresní testování.

6.1.1 Tvorba testů

Všechny testy budou distribuovány spolu s aplikací, aby měl každý uživatel možnost testy využívat. Pro snížení velikosti distribuované aplikace a zachování možnosti měnit libovolně licenci nebude použit žádný framework pro unit testování.

6.1.1.1 Princip

Unit a integrační testy budou fungovat na stejném principu jako např. ve frameworku Nette [9]. Každý test bude vytvářet textový výstup, který bude nejprve ručně kontrolován testerem (unit testování) a následně automaticky porovnáván s referenčním výstupem (regresní testování). Adresář `tests` bude obsahovat adresáře `output` a `ref`. Output bude obsahovat výstupy testů aktuálního testování, `ref` bude obsahovat referenční výstupy. V adresáři `tests` bude shell skript `test.sh`, který spustí všechny testy a zapíše jejich výstupy do adresáře `output`. Následně provede srovnání složky `output` se složkou `ref` a vypíše rozdíly (diff). Vývojář tak velmi rychle získá přehled, zda právě provedenou úpravou nenarušil funkcionalitu aplikace.

6.1.1.2 Formát testového souboru

Každý test bude v samostatném PHP souboru. Na začátku souboru se inicializují potřebné knihovny a načte se testovaná třída. Nad třídou a jejími metodami jsou následně prováděny testy, přičemž se vše vypisuje na výstup. Tester se při psaní testů snaží pokrýt celý rozsah vstupů. Jakmile tester shledá, že testovaná třída vrací na všechny vstupy správné výsledky, zkopíruje výstup do adresáře `ref`. Tím se tento výstup stane referenčním pro další testy (regresní).

6.2 Zhodnocení zvolených řešení

6.2.1 Nepoužití hotového frameworku

První otázka, na kterou bychom se měli zaměřit, je výhodnost použití vlastního frameworku, a to zejména proto, že v dnešní době se psaní vlastního PHP frameworku velmi silně nedoporučuje.

NORS 4 srovnáme se zástupci redakčních systémů postavených nad Zend Framework (Digitalus CMS [2]), CakePHP (Wildflower [37]) a Nette (Joss [14]). Tyto tři frameworky byly vybrány záměrně, protože právě oni byly hlavními kandidáty při výběru základu aplikace.

V zátěžovém testu provnáme parametry spojené se zobrazením úvodní stránky nainstalované aplikace. Množství spotřebované paměti zjistíme PHP funkcí `memory_get_peak_usage` a počet načtených souborů funkcí `get_included_files`. K naměření maximálního počtu požadavků obslužitelných za sekundu použijeme nástroj Apache Bench [6], kde v 10 souběžných vláknech provedeme 1000 GET požadavků. Testovací sestava obsahovala Apache 2.2.9, PHP 5.2.6 s rozšířením APC [25] a MySQL 5.0. PHP bylo spouštěno jako FastCGI aplikace pomocí Apache modulu fcgid [5].

Název	paměť[kB]	načtených souborů	velikost složky[MB]	RPS ¹ [n/s]	RPS s cache[n/s]
Digitalus	2162.72	154	35	6.56	?
Joss	1061.25	28	0.6	26.64	93.34
NORS4	279	33	1.7	155.14	198.03
WildFlower	3742.14	93	23	16.29	66.12

Tabulka 6.1: Srovnání s objektovými WCMS

Z výsledků testu je zřejmé, že po stránce výkonu a spotřeby paměti se vytvoření frameworku na míru opravdu vyplatilo. Velmi překvapivé je, že NORS výkonově velmi výrazně převyšuje i CMS Joss, protože Joss je opravdu značně minimalistický - vůbec nepracuje s databází, nemá administraci a je postaven nad lehkotonážním frameworkem Nette.

¹Requests per second - počet požadavků obslužených za sekundu

6.2.2 Srovnání s nejrozšířenějšími WCMS

V předcházejícím testu jsme srovnali výkonnostní parametry projektu s podobnými projekty postavenými nad známými PHP frameworky. Aby bylo srovnání a hodnocení kompletní, srovnáme NORS také s nejrozšířenějšími redakčními systémy.

Z porovnání vyřadíme projekty PHP-Nuke a phpRS, protože už dva roky neuvádějí žádný vývoj. Porovnány budou opět parametry spojené se zobrazením úvodní stránky nainstalované aplikace. Tentokrát ale nebude načteno PHP rozšíření APC.

Název	paměť[kB]	načtených souborů	RPS[n/s]
Drupal v6.10	6187.47	37	12.11
Joomla! v1.5.10	5692.34	95	10.24
NORS v4.1.0	1014	33	60.58
Nucleus v3.4	2697.35	27	22.75
RS2 v5.3	1147.38	10	57.76
Typo3 v4.2.6	10134.59	36	17.06
WordPress v2.7.1	8298.34	59	8.16

Tabulka 6.2: Srovnání s nejrozšířenějšími WCMS

Z tohoto testu překvapivě vyplývá, že rozdělení kódu do vrstev a používání objektového návrhu nemusí nutně způsobit znatelné zvýšení režie PHP programu. Přestože je NORS 4 postaven na architektuře Model-View-Controller, dokáže se výkonově porovnávat i s RS2, jehož kód je pouze strukturovaný.

6.2.3 Kompaktnost

Jedním z požadavků na projekt byla i kompaktnost výsledné aplikace. Splnění požadavku zjistíme jednoduše srovnáním velikosti archivů redakčních systémů (tar.gz).

Název	velikost archivu[kB]
Joss	133
NORS4	340
Nucleus	525
phpRS	573
RS2	780
Wordpress	881
Drupal	1066
Joomla	1951
PHP-Nuke	3974
Wildflower	5243
Digitalus	6417
Typo3	7872

Tabulka 6.3: Velikost archivu

6.2.4 Efektivita práce s aplikací

Aplikace byla navržena tak, aby tvorba dalších funkcionalit byla co možná nejefektivnější. To je vidět zejména u administrace.

Přidání možnosti spravovat další tabulku do administrace spočívá v těchto krocích:

- vytvoření schématu tabulky (YAML-like soubor)
- přidání položky do menu (1 řádek)
- vytvoření Action Controlleru (3 řádky)

O vytvoření odpovídající tabulky v databázi i vytvoření potřebných tříd Modelu se postará sama aplikace. V administraci se objeví nová položka menu. Po jejím rozkliknutí se zobrazí stránkovatelný výpis tabulky s možností přidávat nové záznamy a editovat a mazat existující. Formulář pro editaci záznamu se sestavuje podle schématu, a proto ho lze kdykoliv upravit.

7 Závěr

7.1 Zhodnocení splnění cílů

Cílem práce bylo vyvinout kompaktní redakční systém určený pro nenáročného uživatele a programátory, přičemž důraz měl být kladen na snadnou rozšiřitelnost, vysoký výkon, snadnou instalaci a konfiguraci, dodržování OOP a přehledné uživatelské rozhraní.

Podle všech testů a srovnání, které byly v předchozí kapitole provedeny, byly cíle práce splněny. Podařilo se vytvořit redakční systém, který má objektový kód a přitom svým výkonem poráží většinu neobjektových konkurentů. NORS se tak stává velmi zajímavou alternativou k dnes široce rozšířeným CMS projektům. Protože byla velká pozornost věnována přehlednosti a rozšiřitelnosti kódu, je NORS velmi vhodný také jako základ pro webové aplikace na míru.

7.2 Další pokračování projektu

Projekt NORS samozřejmě odevzdáním bakalářské práce nekončí, spíše naopak začíná. Nadále bude pokračovat hledání chyb, pokrývání kódu testy, dokumentování a doladování administrace pro laické uživatele.

Až bude projekt zhodnocen jako dostatečně vyspělý a konkurenceschopný, začne fáze propagační.

8 Literatura

- [1] S. Burbeck. Applications programming in smalltalk-80: How to use model-view-controller.
<http://www.math.rsu.ru/smalltalk/gui/mvc.pdf>.
- [2] Digitalus. Digitalus cms.
<http://digitaluscms.com/>.
- [3] Drupal.
<http://drupal.org/>.
- [4] J. Eichorn. phpdocumentor.
<http://www.phpdoc.org/>.
- [5] FastCGI. Mod_fcgid.
<http://fastcgi.coremail.cn/>.
- [6] A. S. Foundation. Apache http server benchmarking tool.
<http://httpd.apache.org/docs/2.2/programs/ab.html>.
- [7] A. S. Foundation. Apache module mod_rewrite.
http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html.
- [8] A. S. Foundation. Jmeter - apache jmeter.
<http://jakarta.apache.org/jmeter/>.
- [9] N. Foundation. Nette framework.
<http://nettephp.com/cs/>.
- [10] GoPHP.
<http://gophp5.org/>.
- [11] P. Group. Image processing (gd).
<http://cz.php.net/gd>.
- [12] P. Group. List of php.ini directives.
<http://www.php.net/manual/en/ini.list.php>.
- [13] D. Grudl. Mvc & mvp.
<http://zdrojak.root.cz/clanky/nette-framework-mvc--mvp/>.
- [14] J. Javorek. joss-cms.
<http://code.google.com/p/joss-cms/>.
- [15] I. N. Jim Arlow. *UML 2 a unifikovaný proces vývoje aplikací*. Computer Press, 2007.
- [16] Joomla.
<http://joomla.org/gf/project/joomla/>.
- [17] J. Miller. Convention over configuration.
<http://msdn.microsoft.com/en-us/magazine/dd419655.aspx>.
- [18] Nucleus.
<http://nucleuscms.org/>.
- [19] C. J. Patterns. Dispatcher view.
<http://www.corej2eepatterns.com/Patterns2ndEd/DispatcherView.htm>.

- [20] R. Pecinovský. *Návrhové vzory*. Computer Press, 2007.
- [21] PHP-Nuke.
<http://phpnuke.org/>.
- [22] phpRS.
<http://www.supersvet.cz/phprs/>.
- [23] T. Reenskaug. The original mvc reports.
http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf.
- [24] RS3. <http://rs.reality-show.net/>.
- [25] G. Schlossnagle. *Pokročilé programování v PHP 5*. Zoner Press, 2004.
- [26] J. Stump. Understanding mvc in php.
http://www.oreillynet.com/pub/a/php/2005/09/15/mvc_intro.html.
- [27] I. Sun Microsystems. Front controller.
<http://java.sun.com/blueprints/patterns/FrontController.html>.
- [28] I. J. Tichý. Architektura aplikace.
<http://www.jantichy.cz/diplomka/poradavky/architektura>.
- [29] Typo3.
<http://typo3.org/>.
- [30] Wikipedia. Active record pattern.
http://en.wikipedia.org/wiki/Active_record_pattern.
- [31] Wikipedia. Business logic.
http://en.wikipedia.org/wiki/Business_logic.
- [32] Wikipedia. Cross-site request forgery.
http://en.wikipedia.org/wiki/Cross-site_request_forgery.
- [33] Wikipedia. Cross-site scripting.
http://http://en.wikipedia.org/wiki/Cross-site_scripting.
- [34] Wikipedia. Indent style.
http://en.wikipedia.org/wiki/Indent_style#BSD_KNF_style.
- [35] Wikipedia. Scaffold (programming).
[http://en.wikipedia.org/wiki/Scaffold_\(programming\)](http://en.wikipedia.org/wiki/Scaffold_(programming)).
- [36] Wikipedia. Systém pro správu obsahu.
http://cs.wikipedia.org/wiki/Systém_pro_správu_obsahu.
- [37] Wildflower. Wildflower.
<http://wf.klevo.sk/>.
- [38] Wordpress.
<http://wordpress.org/>.

9 Přílohy

9.1 Seznam použitých zkratk

AJAX Asynchronous JavaScript and XML

APC Alternative PHP Cache

ASP Active Server Pages

BOM Byte-order Mark

BSD Berkeley Software Distribution

CGI Common Gateway Interface

CMS Content Management System

CSRF Cross-site Request Forgery

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

KNF Kernel Normal Form

MVC Model-View-Controller

OOP Object-oriented Programming

PHP PHP: Hypertext Preprocessor

SQL Structured Query Language

URL Uniform Resource Locator

UTF UCS Transformation Format

WCMS Web Content Management System

WYSIWYG What You See Is What You Get

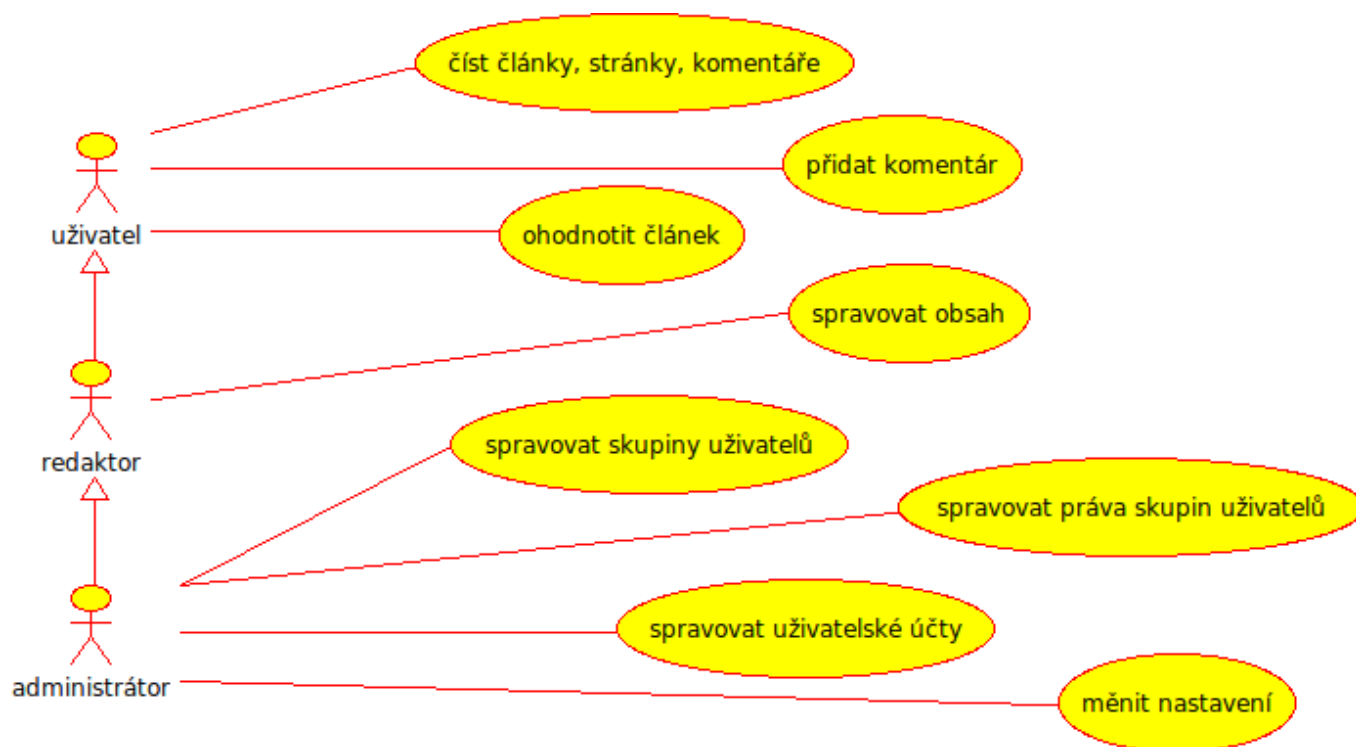
XHTML Extensible HyperText Markup Language

XML Extensible Markup Language

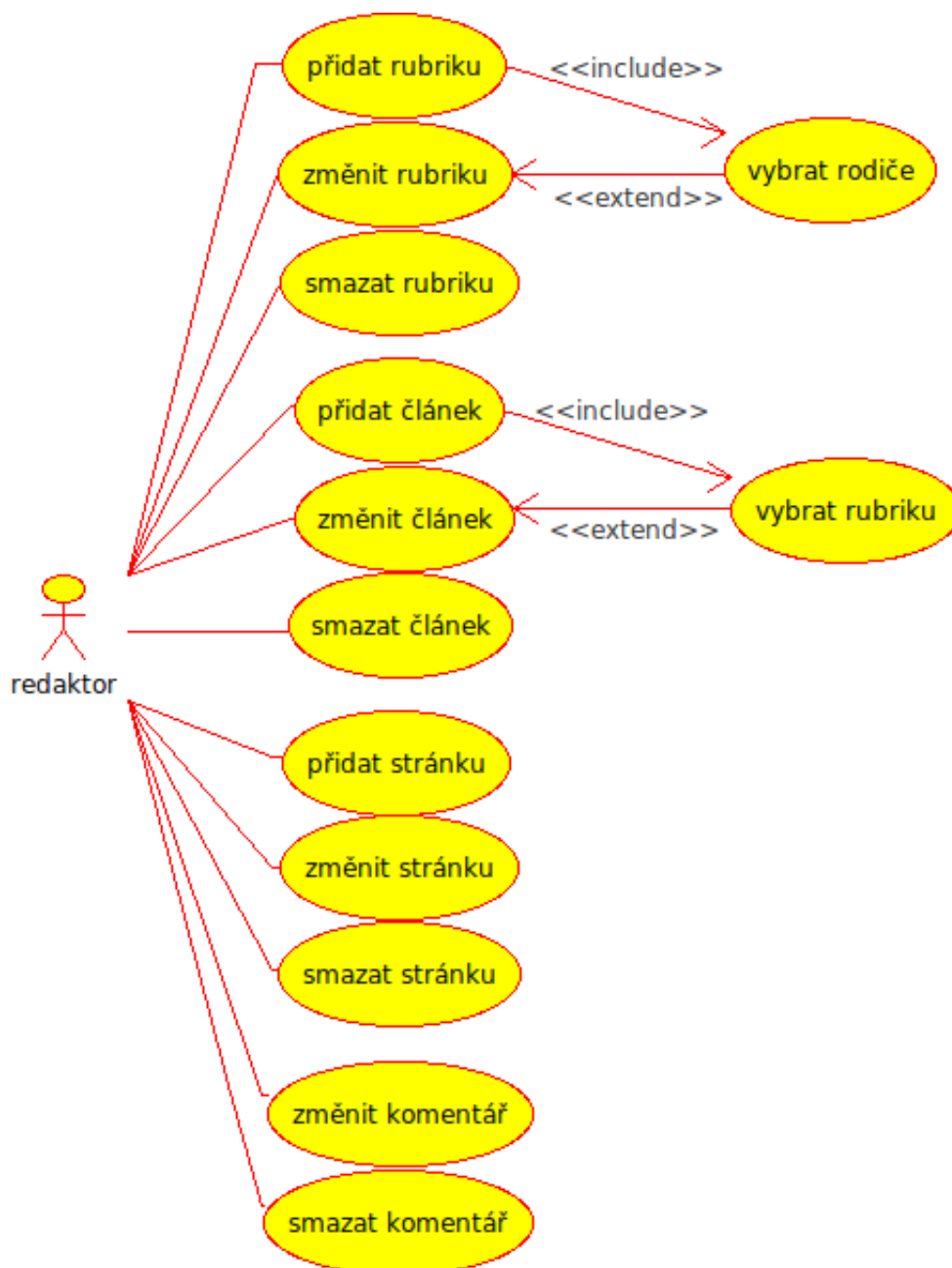
XSS Cross-Site Scripting

YAML YAML Ain't Markup Language

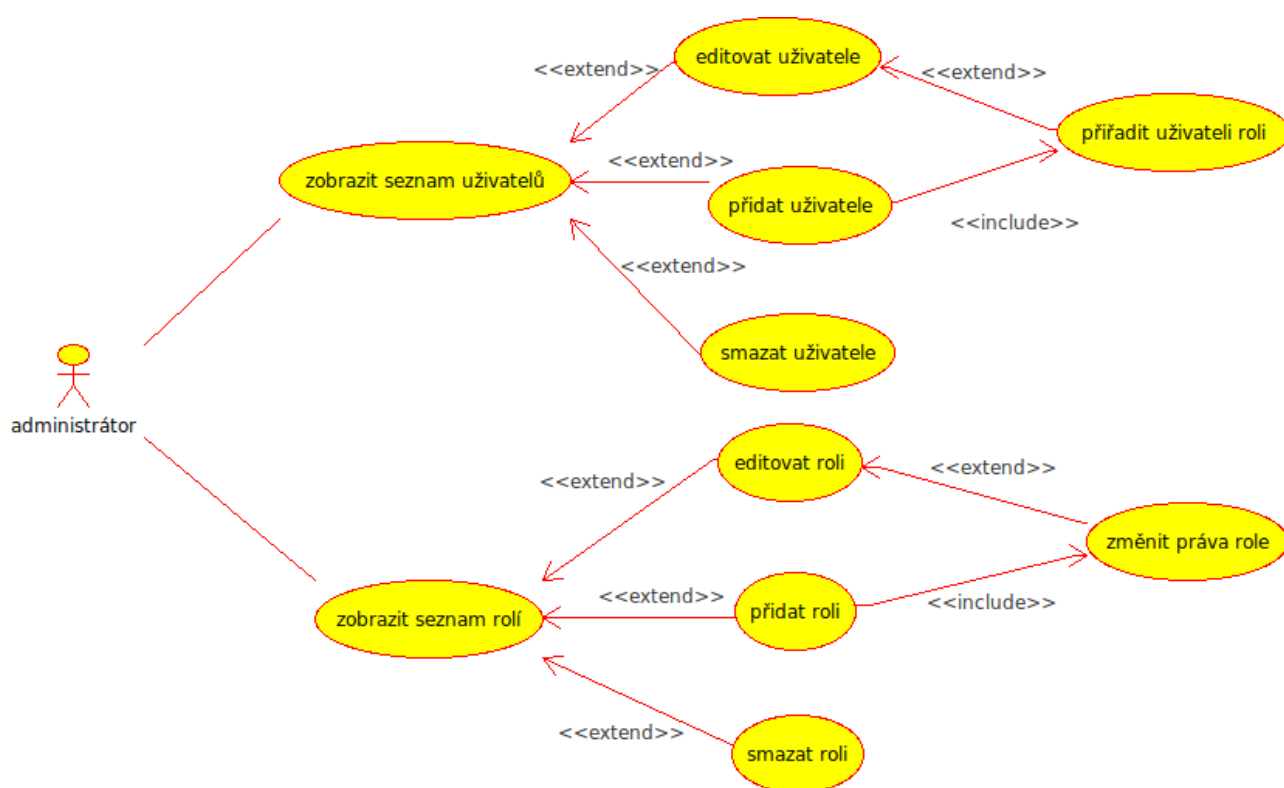
9.2 Use case diagramy



Obrázek 9.1: Celkový use case diagram

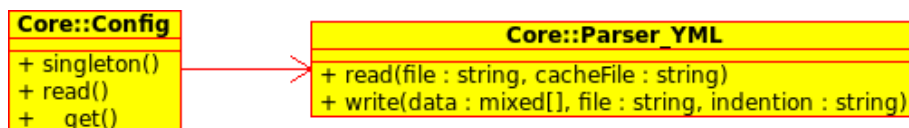


Obrázek 9.2: Správa obsahu

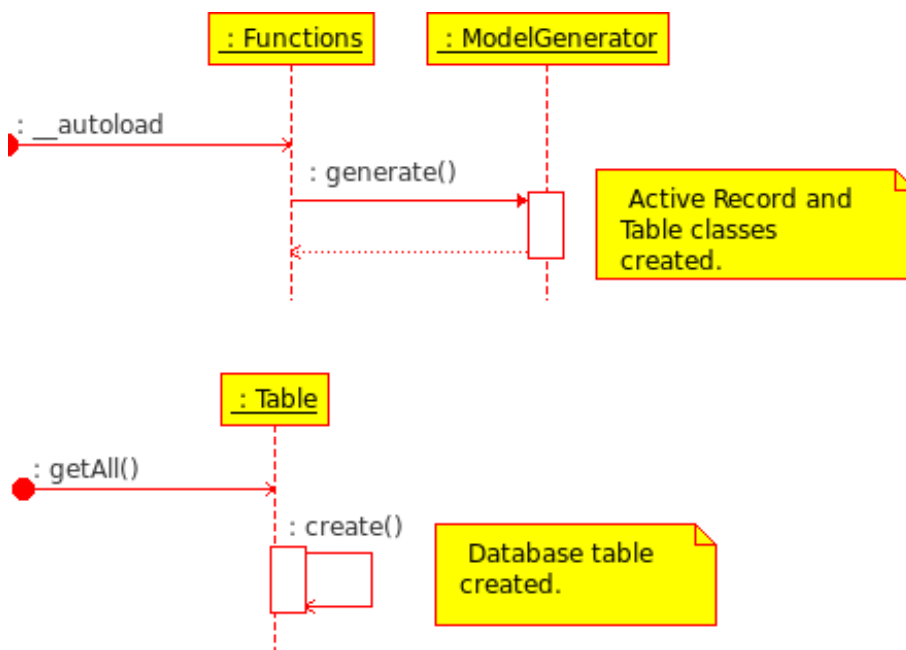


Obrázek 9.3: Správa uživatelů

9.3 Návrhové diagramy a obrázky



Obrázek 9.4: Třídy pro práci s konfigurací



Obrázek 9.5: Průběh scaffoldingu

```

table: post
fields:
  -id_post: int
  +name: string
  -id_user: table
  id_cathegory: table
  text: html
  date: datetime
  active: bool
  karma: double
  -seen: int
  photo: file
ids: id_post
indexes:
  active_date: active, date
  cathegory: id_cathegory

```

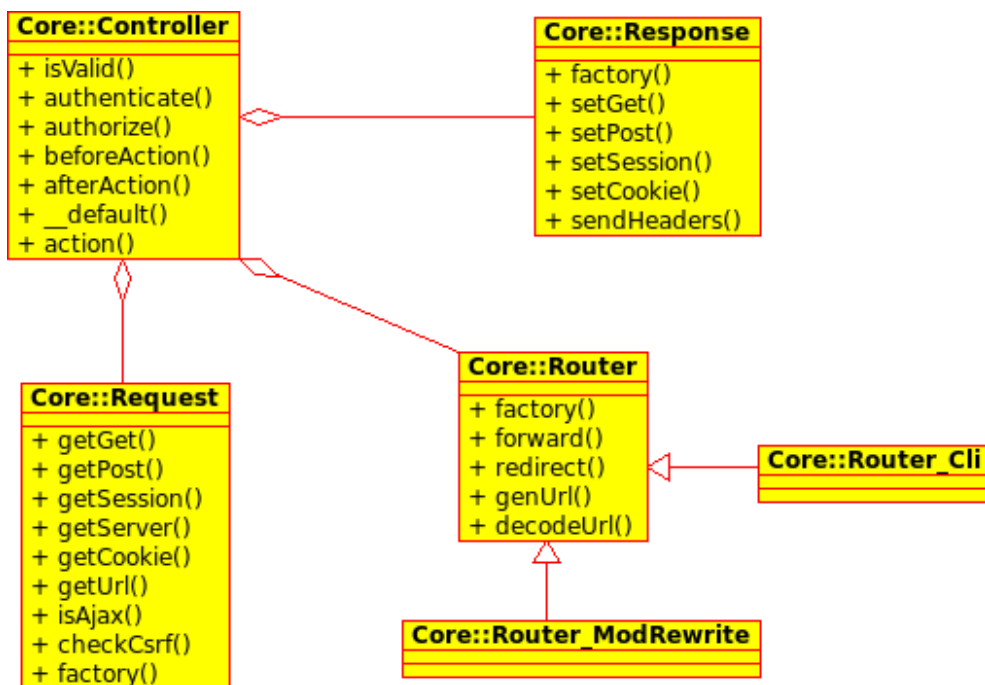
➔

```

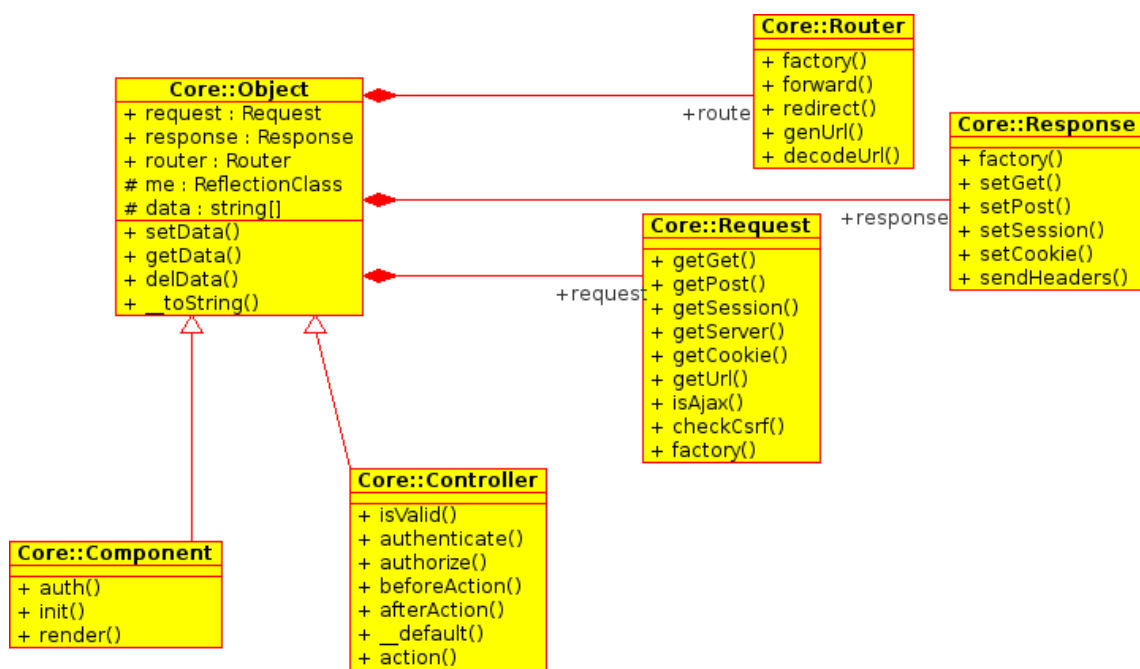
/**
 * ActiveRecord_Post
 *
 * @author Daniel Milde <daniel@milde.cz>
 * @package Core
 */
class ActiveRecord_Post extends Core_ActiveRecord
{
    public function __construct($id = false)
    {
        parent::__construct('post', $id);
    }
}

```

Obrázek 9.6: Scaffolding



Obrázek 9.7: Routing



Obrázek 9.8: Object

