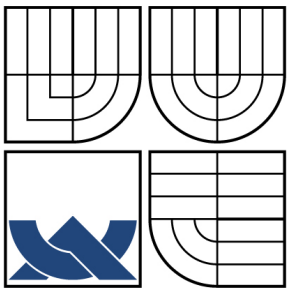
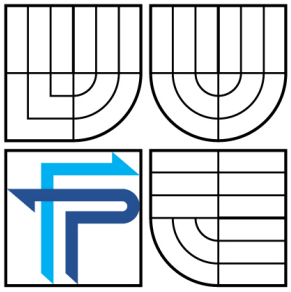


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA PODNIKATELSKÁ
ÚSTAV INFORMATIKY

FACULTY OF BUSINESS AND MANAGEMENT
INSTITUTE OF INFORMATICS

PREZENTACE NA INTERNETU A REDAKČNÍ SYSTÉMY

INTERNET PRESENTATIONS AND CONTENT MANAGEMENT SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PATRIK VRBA
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. MILOŠ KOCH, CSc.
SUPERVISOR

BRNO 2009

Brno University of Technology
Faculty of Business and Management

Academic year: 2008/2009
Institute of Informatics

BACHELOR'S THESIS ASSIGNMENT

Vrba Patrik

Managerial Informatics (6209R021)

Pursuant to Act. No. 111/1998 Coll., on Higher Education Institutions, and in accordance with the Rules for Studies and Examinations of the Brno University of Technology an Dean's Directive on Realization of Bachelor and Master Degree Programs, the director of the Institute of is submitting you a bachelor's thesis of the following title:

Internet Presentations and Content Management Systems

In the Czech language:

Prezentace na internetu a redakční systémy

Instructions:

Pursuant to section 60 of Act No. 121/2000 Coll., as subsequently amended (hereinafter referred to as "Copyright Act") this thesis is a "schoolwork" and its use complies to the legal regime of the Copyright Act. Quotations hereof are permitted upon prior consent of the Faculty of Business and Management of Brno University of Technology. Before any external use of this thesis it is strictly required to conclude a "License Agreement" pursuant to the Copyright Act.

Literature / Sources:

DOMES Martin: Tvorba webových stránek. Computer Press, a. s., 2006. 192 s. ISBN: 80-251-0920-8.
GUTMANS, Andi. Mistrovství v PHP 5 . 1.vyd. Praha: Computer Press, 2005. 520 s. ISBN 80-251-0799-X .
KRČMÁŘ, Jakub. Adobe Photoshop praktický webdesign. 1.vyd. Praha: Grada, 2007. 204 s. ISBN 80-247-1423-X .
WEINMAN, Lynda. Velká kniha webdesignu. 1.vyd. Praha: Zoner Press, 2004. 528 s. ISBN 80-868-1510-2 .

The supervisor of bachelor's thesis: doc. Ing. Miloš Koch, CSc.

Submission deadline bachelor's thesis is given by the Schedule of the Academic year 2008/2009.

L.S.

Ing. Jiří Kříž, Ph.D. Director of the Institute	doc. RNDr. Anna Putnová, Ph.D., MBA Dean of the Faculty

Brno, 26.05.2009

Anotace

Tato práce analyzuje problematiku tvorby internetových prezentací a vývoje jejich redakčních systémů. Na základě zhodnocení dostupných technologií z oblasti internetových stránek a webových aplikací, je následně navrhnut a implementován konkrétní redakční systém pro správu webové prezentace.

Annotation

This thesis covers developing process of internet presentations and their content management systems. Based on evaluation of available technologies from internet sites and web application area is subsequently designed and implemented particular web presentation content management system.

Klíčová slova

Redakční systém, CMS, internet, WWW, SEO, HTML, CSS, Ruby on Rails, relační databáze, Adobe Flex

Keywords

Content management system, CMS, internet, WWW, SEO, HTML, CSS, Ruby on Rails, relation databases, Adobe Flex

Bibliografická citace

VRBA, P. Prezentace na internetu a redakční systémy. Brno : Vysoké učení technické v Brně, 2009. 55 s. Vedoucí bakalářské práce doc. Ing. Miloš Koch, CSc.

Čestné prohlášení

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 25. Května 2009 …….

Podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce doc. Ing. Miloši Kochovi CSc. za hodnotné rady, připomínky a odborné vedení při zpracování této práce.

Obsah

1 Úvod

Internet poskytuje relativně levné řešení, jak celosvětově prezentovat informace. Komunita kolem tohoto média neustále roste, což podnítilo mnoho komerčních podniků k jejich vlastní prezentaci výrobků anebo služeb na této síti. Velký podíl potencionálních zákazníků hledá informace právě zde a z tohoto důvodu podniky, které se neprezentují na internetu, často přicházejí o nové zakázky. Je pravdou, že určité obory podnikání mají na internetu větší potencionál než jiné, přesto umístění alespoň jednoduché vizitky se základními informacemi o podniku, je dnes považováno za nutnost.

Vlastnictví internetové prezentace však s sebou nese určité náklady, které vznikají nejen samotným vytvořením webu, ale v neposlední řadě i při jeho následném provozu. Pro dosažení hodnotné prezentace je zapotřebí zajistit informace, které by měly být neustále aktualizovány tak, aby odrážely skutečný stav. Aktualizace ovšem pro běžného uživatele představuje technickou překážku, kterou má řešit tato práce.

1.1 Cíl práce

Tato práce analyzuje problematiku okolo tvorby webových prezentací a jejich redakčních systémů. Cílem je zhodnotit aktuální situaci a poslední trendy v oblasti webových technologií a na základě této studie nabídnout co možná nejlepší řešení v tvorbě webových prezentací a redakčních systémů. Hlavním výstupem je pak návrh a implementace konkrétního redakčního systému, jehož pomocí bude schopen i laik spravovat obsah webové prezentace.

Je prakticky nemožné vytvořit dokonale univerzální a přitom pro laika snadno ovladatelný systém. Specifické požadavky uživatelů vyžadují implementaci na míru. Proto výsledný systém je navrhovaný s důrazem na modularitu, přičemž tato práce by měla nabídnout vhodný technologický rámec, pomocí kterého lze dosáhnout cílů co nejefektivněji.

1.2 Vymezení problému

Objektem výzkumu je společnost GMV Martini CZ s.r.o., člen italské holdingové společnosti GMV Group, jejímž oborem podnikání je výroba komponent hydraulických

výtahů. Dceřiné společnosti si samy zajišťují svoji propagaci na trhu, což platí i v případě webové prezentace. Zastoupení pro Českou a Slovenskou republiku v podobě již zmíněné společnosti GMV Martini CZ s.r.o. disponuje webovou prezentací, nicméně současné řešení je již nedostatečné. Z tohoto důvodu se společnost poptává po nové prezentaci, jejíž součástí by byl i redakční systém pro správu obsahu.

2 Teoretická východiska práce

2.1 HTML a XHTML

Jazyk HTML, nebo jeho celým názvem HyperText Markup Language, je jazyk pro vytváření tzv. hypertextových dokumentů. Hypertextem se rozumí dokument, jenž obsahuje odkazy na jiné dokumenty, nebo jejich části. Rozsah možností HTML je ovšem širší. Volněji jej lze popsat jako jazyk, který umožňuje tzv. tagy definovat význam jednotlivých částí dokumentu, jako jsou odstavce, nadpisy tabulky apod.

Původní specifikace HTML byla vytvořena pomocí jazyka SGML – Standard Generalized Markup Language, což je jazyk pro definici značkovacích jazyků. Postupem času se ukázalo, že SGML je zbytečně komplexní a jeho úplná implementace velice náročná. Nepříznivou situaci HTML ještě umocňovaly snahy výrobců webových prohlížečů, kteří rozšiřovali možnosti jazyka o nestandardní prvky. Prohlížeče navíc byly benevolentní vůči chybám, které nebyly v souladu se samotnou definicí jazyka. Toto všechno vedlo k situaci, kdy existovaly HTML dokumenty, které dokázal správně interpretovat pouze konkrétní prohlížeč, pro který byl dokument vytvořen.

Odpovědí na zmíněné problémy, bylo vytvoření nového jazyka konsorciem W3C pro definici značkovacích jazyků, který byl pojmenován XML – eXtensible Markup Language. Pomocí XML pak byl vytvořen jazyk XHTML, který odstraňoval nekonzistentnosti mezi výrobci prohlížečů. XHTML se vrací k původní myšlence HTML, tedy popisovat strukturu dokumentu, ne jeho vizuální nebo jinou prezentaci. Byly proto odstraněny značky určené k definici grafické podoby a tato úloha byla přenechána technologii CSS. Schematické chyby již nadále nejsou tolerovány, čímž bylo docíleno přenositelnosti. V průběhu doby XHTML získalo silnou pozici na webu a dnes již lze říci, že nahradilo staré nekompatibilní HTML od výrobců prohlížečů. Zpracováno podle [11].

2.2 XML

eXtensible Markup Language je značkovací metajazyk určený k popisu strukturovaných dat, popřípadě k definici nových značkovacích jazyků, jehož specifikaci vytvořilo W3C konsorcium. Tento jazyk sám o sobě neřeší problematiku prezentace dokumentu, ale pouze popis významu jeho jednotlivých částí. Pro definici grafické prezentace obsahu

může být použito např. kaskádových stylů, popřípadě dokument může být přetransformován pomocí XSLT pravidel do nového dokumentu.

Vzhledem k tomu, že specifikace formátu je volně dostupná a není svázána s žádnou konkrétní platformou, XML je dnes široce zastoupeno v mnoha oblastech, kde je potřeba výměnného formátu pro popis strukturovaných dat. Jeho aplikace tak lze nalézt mezi souborovými formáty, síťovými komunikačními protokoly, ale třeba také mezi programovacími jazyky.

2.3 CSS

Cascading Style Sheets, česky označovaný jako kaskádové styly, je jazyk užívaný při definici prezentace dokumentu, který je popsán v určitém značkovacím jazyce. Pro jeho aplikaci se nemusí přímo jednat o HTML dokument, nicméně právě díky rozšířenosti HTML je CSS dnes tak populární.

Za jeho vznikem stojí především snaha oddělit samotný obsah a strukturu dokumentu, od jeho prezentace, tím se zlepší přístupnost samotného obsahu a řízení jeho prezentace v jakékoliv podobě. Specifikace CSS kromě části pro grafický návrh zahrnuje i pravidla pro prezentaci na jiném, než grafickém zařízení. Může se jednat např. o audio prezentace pro nevidomé apod.

Jelikož obsahová část je oddělena od pravidel pro prezentaci, je možné podle potřeby na dokument aplikovat různé CSS styly. Tímto způsobem je možné definovat odlišné způsoby prezentace např. pro monitor, projektor, mobilní zařízení, tiskárnu apod., bez nutnosti zásahu do samotného dokumentu. Zpracováno podle [14].

2.4 JavaScript

JavaScript je skriptovací programovací jazyk, především využívaný při tvorbě dynamických webových stránek. Přestože existují technologie, které podporují JavaScript na straně serveru, jeho hlavní využití spočívá v možnosti vykonávat jeho skripty na klientské straně. Byl vyvinut společností Netscape a ve spojitosti se společností Sun Microsystems, byl zvolen jeho název využívající obchodní úspěšnost jazyka Java. JavaScript se rychle stal populárním nástrojem, k oživení webových stránek a tak společnost Microsoft pro svůj konkurenční prohlížeč vytvořila vlastní implementaci označenou jako JScript. Tato implementace ovšem nebyla kompatibilní

s JavaScriptem, proto byl jazyk standardizován organizací Ecma International a standardizovaná verze byla označena jako ECMAScript. Z ní byly odvozeny i další implementace jako např. ActionScript využívaný ve Flashi.

Podpora JavaScriptu ve webových prohlížečích nabyla nového rozměru se vznikem AJAXu – Asynchronous JavaScript and XML. AJAX je označení pro technologie umožňující tvorbu interaktivních webových aplikací, které na pozadí komunikují se serverem a mění obsah bez nutnosti opětovného načtení celé stránky. Uživatel tak má pocit větší plynulosti, která se blíží klasickým desktopovým aplikacím. Zpracováno podle [4].

2.5 Rozhraní CGI

Common Gateway Interface nebo zkráceně jen CGI je technologie, která stála na počátku dynamicky generovaných webových stránek. Jedná se o specifikaci rozhraní definující, jakým způsobem předávají prohlížeče informace zpět serveru, který poté generuje dynamický obsah na základě těchto informací. Klíčovou částí je tzv. CGI skript, který je spuštěn webovým serverem ve chvíli kdy přijde požadavek od klienta. Informace od klienta jsou CGI skriptu předány v závislosti na HTTP metodě a nastavení webového serveru, jako parametry příkazové řádky, proměnné prostřední nebo na standardním vstupu. Díky jednoduchosti rozhraní je možné vytvářet CGI skripty v libovolném programovacím jazyku, ovšem nejčastěji jsou CGI skripty vytvářeny v Perlu, C, Pythonu nebo shellu, v případě použití Unixu.

Hojné používání rozhraní CGI ve své době s sebou přineslo mnoho bezpečnostních problémů. V případě, že obslužný program má povoleno spouštět jiné programy na serveru, mohou někteří lidé najít cesty, jak spustit nežádoucí programy, nežádoucím způsobem.

V dnešní době se tvorba CGI skriptů považuje za překonanou. CGI poskytuje pouze holé komunikační rozhraní s webovým serverem a programátor CGI skriptu se musí sám vypořádat se všemi záludnostmi při generování odpovědi. Přesto dodnes je tato technologie využívána, a to sice jako možná alternativa pro propojení serveru s webovým frameworkem.

2.6 PHP

PHP (z angl. PHP: Hypertext Preprocessor) je skriptovací programovací jazyk, vytvořený především pro tvorbu dynamických webových stránek. PHP skripty jsou do značné míry nezávislé na platformě a mohou být přeneseny na různé operační systémy. Skripty jsou vykonávány na straně serveru a klientovi je přenášen jejich výsledek.

PHP je velice populárním nástrojem pro tvorbu dynamických webů. Je to především způsobeno jeho jednoduchostí. Samotný kód lze vkládat přímo do webových stránek, což je zejména pro začínající programátory výhodné, jelikož se nemusí zabývat architekturou aplikace. V případě větších aplikací, absence určité programové struktury může způsobit velké problémy při úpravách, resp. požadavku rozšíření aplikace. V průběhu doby vzniklo mnoho frameworků využívajících PHP, které do značné míry poskytují možnost vytvářet strukturované aplikace. Přesto PHP dosud trpí problémy, které brání této technologii proniknout na pole větších systémů. PHP se vyvíjí živelně, jazyk není formálně popsán, pouze existuje jeho implementace, knihovna funkcí obsahuje nekonzistentní signatury funkcí, každý požadavek je zpracován v novém kontextu, což snižuje výkon a může způsobit problémy v konkurenčním zpracování požadavků.

Dlužno podotknout, že PHP, jak již napovídá jeho původní název – Personal Home Page – byl vytvořen za účelem tvorby jednoduchých dynamických webů, proto na něj nelze klást požadavky kladené na velké nebo dokonce kritické systémy. PHP je dnes velice rozšířené, na trhu je k dispozici nespočet hostingů, jeho popularita zapříčinila vznik velkého množství nejrůznějších programových knihoven a tak, pro tvorbu jednodušších webů se nemusí jednat o špatnou volbu, v některých systémech je např. PHP použito pouze pro prezentační vrstvu.

2.7 Relační databáze

Relační databáze byla představena již v roce 1970, jejím autorem dr. Edgarem F. Coddem, který v té době pracoval jako výzkumný pracovník u společnosti IBM. Tento druh databází vychází z matematické teorie množin a logických predikátů prvního řádu.

Základem relačních databází jsou tabulky, jejichž sloupce mají určen svůj datový typ. Řádky potom představují jednotlivé záznamy uložených dat. Představíme-li si kartézský

součin všech přípustných hodnot jednotlivých atributů tabulky, pak konkrétní tabulka v databázi představuje podmnožinu takového součinu. V matematické terminologii se hovoří o relaci, z které vzniklo pojmenování pro tento typ databází.

Důležitou částí relačních databází jsou primární a cizí klíče. Primární klíč představuje unikátní identifikátor záznamu, který může být tvořen jedním, nebo více atributy tabulky a musí být vždy určený. Cizím klíčem lze mezi databázovými tabulkami vytvářet vztahy a určit tak, které záznamy v databázi mezi sebou souvisí.

Hodnota databáze se odvíjí od kvality jejích dat. K zamezení zjevně chybných údajů se využívá integritních omezení, která zabezpečují, aby databáze byla v konzistentním stavu, tedy aby splňovala definovaná integritní omezení. Integrita databáze v sobě zahrnuje několik druhů omezení:

- *Entitní* – vyžaduje úplnost primárního klíče
- *Doménové* – dohlíží na dodržování definovaných datových typů
- *Referenční* – kontroluje platnost relačních vazeb
- *Aktivní* – jenž definuje reakci databáze v případě, že by mělo dojít k porušení referenční integrity

Relační vazby vyskytující se v databázi mezi tabulkami, mohou vznikat v jedné ze tří forem a to 1:1, 1:N a N:M. Tato označení reflektují, kolik záznamů v dceřiné tabulce patří, nebo může patřit záznamům v rodičovské tabulce. Použitím těchto relačních vazeb se mimo jiné zabrání redundanci dat v databázi.

2.7.1 MySQL

MySQL je systém pro řízení báze dat, určený pro práci s relačními databázemi. Popularita tohoto systému je obrovská, v současnosti existuje více jak 11 miliónů instalací. Na druhé straně MySQL doposud nepodporuje mnoho vlastností, které jsou dostupné v jiných běžných databázových systémech. Absence těchto vlastností ovšem zjednodušuje instalaci a následnou správu databáze, což je jeden z hlavích důvodů dnes tak velké popularity tohoto produktu.

MySQL je systém založený na architektuře klient/server, což umožňuje, aby k jednomu databázovému serveru přistupovalo libovolné množství klientů, kterými jsou nejčastěji

aplikační programy. Tato architektura je víceméně běžná u všech populárních databázových systémů.

Jak již bylo nastíněno výše, MySQL je relativně jednoduchá databáze, ve které chybí celá řada vlastností, jinak dostupných u jiných, především komerčních systémů. Od verze 5 je podporováno větší množství funkcí, které ovšem nejsou zcela hotové. Nespornou výhodou MySQL je skutečnost, že se jedná o bezplatně distribuovaný open source projekt, který je podporovaný na mnoha operačních systémech a velice rozšířený mezi poskytovateli hostingů. Zpracováno podle [7].

2.8 Ruby on Rails

Ruby on Rails je relativně nový webový framework, vytvořený v programovacím jazyce Ruby. Jedná se o open source projekt uplatňující metodologii agilního vývoje. Podobně, jako mnoho jiných moderních webových frameworků, v Railsech se využívá MVC (Model-View-Controller) architektury. Aplikace je tak rozdělena do tří vrstev na datový model, poskytující funkce pro přístup a manipulaci s daty; řadič, který je zodpovědný za funkční stránku věci; a pohled, jehož úkolem je prezentace dat získaných od řadiče.

Programovací jazyk Ruby je ryze objektově orientovaný programovací jazyk, naprosto vše zde vystupuje jako objekt. K dispozici je mnoho dynamických vlastností, pro práci s prvky programovacího jazyka. Dynamičnost je na tak vysoké úrovni, že je jazyk Ruby mnohdy zatracován právě kvůli ní. Je např. možné za běhu měnit definice tříd nebo jednotlivých instancí, měnit implementace v základních třídách jazyka apod. Nicméně při vhodné aplikaci těchto vlastností jazyka lze docílit velice šikovných konstrukcí, které ve svém důsledku zvyšují produktivitu programátora. Živým důkazem tohoto tvrzení je právě Ruby on Rails. Zpracováno podle [15].

2.8.1 Filosofie

V Ruby on Rails se důsledně uplatňují pravidla *Convention over Configuration* – CoC a *Don't Repeat Yourself* – DRY. Cílem těchto pravidel a principů je zvýšení produktivity ve fázi vývoje a to tím způsobem, aby vývojáři neztráceli čas u činností, které nejsou nezbytné. V tuto chvíli se zmíněný výrok může zdát prázdný a jeho tvrzení samozřejmé,

nicméně vysvětlení významu nechám na pozdější části této práce, kde bude ukázáno, jak takových cílů je dosaženo.

V případě samotné agilní metodologie [1]se staví do popředí rychlost vývoje a to i na úkor klasických pouček softwarových inženýrů. Snižuje se význam analýzy a dokumentace a za nejzákladnější dokumentaci je považován samotný zdrojový kód. To zvyšuje pozornost, při jeho tvorbě a je tak nutné více dbát na jeho kvalitu, neboť se předpokládá, že se k němu bude více vracet a bude nutné jej rychle pochopit a rychle modifikovat. Součástí agilního vývoje je větší počet krátkých iterací zadání/vývoj/testování, který tak umožňuje plynuleji komunikovat se zadavatelem projektu a tak rychleji a přesněji dosáhnout konečného cíle. Samotný framework Ruby on Rails samozřejmě nevyžaduje aplikaci agilních principů, ale poskytuje pro ně vhodné zázemí.

2.8.2 Architektura MVC

Architektura Model-View-Controller je návrhový vzor, určený k tvorbě interaktivních aplikací. Vyvíjená aplikace je tak rozdělena do tří komponent, a to sice na datový model, řadič a prezentační vrstvu.

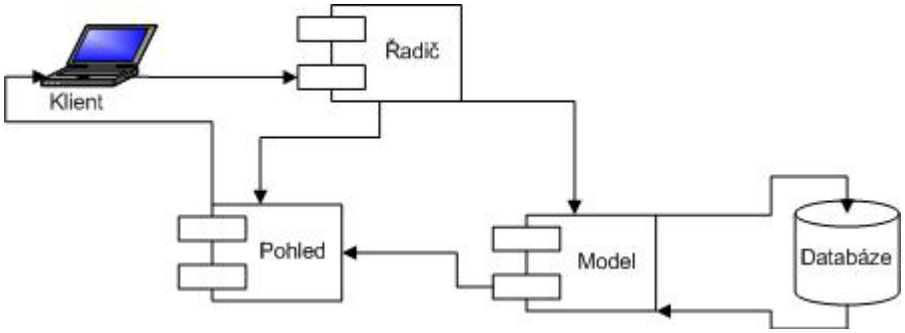
V nejširším úhlu pohledu, datový model je zodpovědný za udržování stavu aplikace. Tento stav může být jak dočasný, tak trvalý. Příklad dočasného stavu je např. příznak, že uživatel je přihlášený. Trvalý stav aplikace je uložen mimo samotnou aplikaci, např. v databázi. Důležité je, že datový model neplní jen funkci přístupu k datům, ale zároveň obsahuje aplikační logiku, která se skládá z pravidel, podle kterých může být s daty zacházeno. Určité údaje např. mohou být povinné, nebo musí splňovat určité pravidlo formátu. Vzhledem k tomu, že datový model sám o sobě obsahuje tato pravidla, plní tak funkci určitého ochránce dat a žádná jiná část aplikace nemůže zapříčinit poškození datového úložiště.

Pohled, nazývaný také jako prezentační vrstva, je zodpovědná za vytváření uživatelského rozhraní a to většinou na základě dat z datového modelu. Pohled neobsahuje žádnou funkčnost, pouze slouží jako zprostředkovatel mezi aplikací a uživatelem. Oddělení této úlohy do samostatné komponenty umožňuje na základě

[1] VACULNÝ, Jiří. *Agilní metodologie programování*. [online] [cit. 2009-05-08] Dostupné z: http://www.fi.muni.cz/usr/jkucera/pv109/2003/xvaculny.htm.

jedněch dat vytvářet různé výstupy, podle použitého pohledu. V neposlední řadě, úpravy provedené v aplikaci nemusí vést k zásahům do uživatelského rozhraní a naopak.

Poslední částí je řadič, který vytváří mezistupeň mezi modelem a pohledem a tak v konečném důsledku řídí chod celé aplikace. Činnost řadiče začíná ve chvíli, kdy obdrží událost (např. uživatelský vstup). Podle typu události je vybrána adekvátní akce, ve které řadič komunikuje s datovým modelem a nakonec vybere příslušný pohled, jenž zobrazí.



Obrázek 1- schéma MVC modelu

V kontextu webové aplikace, postavené na Ruby on Rails frameworku, scénář začíná ve chvíli, kdy na server dojde HTTP požadavek. V prvním kroku dojde k analýze URL adresy požadované stránky, jejímž výsledkem je určení řadiče a jeho akce, která bude na požadavek odpovídat. Po spuštění patřičné akce dojde ke komunikaci s datovou vrstvou, která zpřístupní požadovaná data umístěná v databázi. Tato data jsou následně odeslána do prezentační vrstvy, kde dojde k formátování do konečné podoby a formátu, který si vyžádal klient. Výsledek činnosti pohledu je nakonec odeslán webovým serverem. Zpracováno podle [16, str. 22 – 25].

2.8.3 Datový model

Obecně webové aplikace udržují svoje data v relačních databázích. I v případě, že se jedná o málo strukturovaná data, jako je samotný text, relační databáze poskytují víceméně standardizovaný a snadný způsob pro jejich ukládání, hledání, řazení apod. Přesto relační databáze, vycházející z matematické teorie množin, jsou těžko použitelné v kontextu objektově orientovaných programovacích jazyků. Objekty obsahují vše o datech a operacích nad nimi, zatímco databáze obsahují pouze samotná data

(pomineme-li v tuto chvíli pokročilejší techniky jako jsou uložené procedury apod.). Aplikace napsané v některém z procedurálních programovacích jazyků v sobě nejčastěji obsahují přímo vložené SQL příkazy, které jsou odesílány na databázi. Tento způsob je přímočarý a často používaný a může vypadat ideálně pro jednoduché aplikace. Nicméně mícháním aplikační logiky s přístupem k datům v databázi může vést k těžkostem se správou a rozšiřováním aplikace v budoucnosti. Jestliže je potřeba provést určitou operaci s daty, požadovaný kód se vloží přímo do místa, kde tato potřeba vznikla. Později se ovšem může ukázat, že daná operace je potřebná i na jiných místech, čímž vznikne duplicitní kód. Pozdější úpravy pak vyžadují nalézt všechna tato místa a upravit je, což mimo jiné velice zvyšuje rizikovost vzniku nových chyb v aplikaci.

V objektově orientovaném programování je toto řešeno tzv. zapouzdřením. Vše co se vztahuje k určitému objektu v aplikaci, je umístěno v jeho třídě, která definuje rozhraní, pomocí kterého je možné vykonávat dané operace, resp. přistupovat k jeho datům. Samotná implementace je tak na jednom místě a je ukryta za rozhraním před zbytkem světa, což znemožňuje nepovolené zásahy a přístupy do objektu, díky čemuž je zbytek aplikace závislý výlučně na rozhraní a ne na vnitřní implementaci.

Tyto zásady byly převzaty do oblasti programování databází. Základním konceptem je zapouzdřit přístup k datům do tříd, se kterými pak pracuje zbytek aplikace, na rozdíl od přímého přístupu k databázovému schématu. Vytvoření takových tříd ovšem není triviální záležitostí, vzhledem k tomu, že taková datová vrstva musí dohlížet na životní cyklus objektů v aplikaci a entit v databázi a mezi entitami v databázi existují relační vazby, které by měly být přístupné z aplikace apod. Je tedy vhodné použít některou, již vytvořenou tzv. ORM (Object-Relational Mapping) knihovnu, namísto snahy znovu vymýšlet již objevené.

Ruby on Rails je standardně dodávaný s ORM knihovnou ActiveRecord. Tato knihovna umožňuje mapovat databázové objekty do aplikační vrstvy běžným způsobem, jak tomu bývá u podobně zaměřených knihoven. Tabulky v databázi jsou mapovány na třídy, jednotlivé záznamy tabulky pak na objekty dané třídy. Tyto objekty pak obsahují atributy, které jsou použity k přístupu nebo nastavení hodnoty jednotlivých sloupců. V neposlední řadě, knihovna ActiveRecord poskytuje celou řadu pomocných metod. Tyto metody lze rozdělit do dvou kategorií a to sice na metody tříd, což jsou operace na

úrovni tabulky a metody objektů, které zastupují operace na úrovni záznamu. Kupříkladu, metody sloužící k hledání záznamů jsou obsaženy v definici třídy, zatímco metody pro vytvoření nebo aktualizaci záznamu jsou součástí objektu.

Typická ORM knihovna při inicializaci vyžaduje, aby jí byla předána konfigurace, ve které je určeno, jakým způsobem jsou namapovány entity v databázi na její protějšky v aplikaci. V případě knihovny ActiveRecord toto není vyžadováno a místo toho, jsou definovány konvence, které určují implicitní hodnoty. Kompletní namapování třídy do databáze pak může vypadat následovně:

```
class Job < ActiveRecord::Base
end
```

Na úrovni programovacího jazyka Ruby se jedná o definici třídy *Job*, která dědí ze třídy *Base* umístěné v modulu *ActiveRecord*. Veškerou práci zajistí rodičovská třída *Base*, která využívá dynamických možností jazyka Ruby. Při inicializaci této namapované třídy, je nejprve zjištěn název třídy, který je převeden na malá písmena a do množného tvaru, přičemž jednotlivá slova v názvu, jsou oddělena podtržítkem. Takto upravený název třídy (v tomto případě *jobs)*, je pak použit jako název podkladové tabulky v databázi. ActiveRecord poté odešle do databáze dotaz, kterým zjistí, jaké sloupce tato tabulka obsahuje. Na základě výsledku dotazu, jsou do definice třídy přidány atributy objektu, se stejnými názvy jako jsou názvy sloupců v tabulce a které umožňují manipulaci s hodnotami sloupců.

Z rodičovské třídy *Base* se dále zdědí celá řada metod, pro práci s danou tabulkou, jako jsou metody pro hledání, přidávání nebo aktualizace záznamů. Mezi těmito metodami lze najít všechny potřebné operace při práci s datovým modelem a tak jen podotknu, že výše zmíněný příklad opravdu představuje funkční namapování databázové tabulky na její protějšek v aplikaci.

Pro relační vazby v databázi, lze použít několik deklarací podle typu a strany vazby, vzhledem k mapované třídě. I zde existují konvence, pomocí kterých je ORM knihovna schopna si domyslet konkrétní hodnoty. Pro sloupec cizího klíče se implicitně použije název rodičovské tabulky v jednotném čísle, následovaný řetězcem „_id“. V případě vazby typu N:N se jako průniková entita hledá tabulka s názvy obou tabulek, seřazených v abecedním pořadí, mezi nimiž je znak podtržítka. Konkrétně se jedná o

deklarace *belongs_to*, na straně cizího klíče; *has_one* nebo *has_many* na straně rodičovské tabulky a *has_and_belongs_to_many* na obou stranách vazby typu N:N. Budeme-li uvažovat dříve zmíněný příklad, kde každá pracovní pozice bude umístěna v jedné kategorii, tedy tabulka *jobs* bude obsahovat cizí klíč, namapované třídy budou vypadat následovně:

```
class JobCategory < AcriveRecord::Base
  has_many :jobs
end

class Job < ActiveRecord::Base
  belongs_to :job_category
end
```

Knihovna ActiveRecord nabízí mnoho dalších možností. Jak již bylo řečeno, požadavkem na datový model není jen přístup k datům v databázi, ale také aby obsahoval aplikační logiku, která zabrání poškození dat. K tomuto účelu jsou k dispozici nejrůznější validace, které nejen zabrání nepovolené operaci nad daty, ale také poskytnou patřičné chybové hlášení, jenž je integrované se zbytkem Ruby on Rails frameworku a tak jej lze pomocí jediného příkazu zobrazit na patřičném místě v prezentační vrstvě aplikace. Pro názornost uvedu rozšířený model *Job*, který obsahuje textový sloupec *description*, jehož hodnota by měla být dlouhá 3 až 10 000 znaků.

```
class Job < ActiveRecord::Base
  belongs_to :job_category

  validates_length_of :description, :within => 3..10_000
end
```

Pro datové modely je k dispozici celá řada validací, od těch jednodušších, jako je výše zmíněný příklad validace délky řetězce, přes flexibilnější varianty, kterou je např. validace proti regulárnímu výrazu, až po vytváření vlastních validací, nejčastěji skrze přetížení metody *validate*. Podstatnou skutečností je, že jakmile má dojít k uložení hodnot objektu do databáze, tyto validace jsou vždy vykonány a v případě, že některá validace selže, zápis hodnot do databáze nebude uskutečněn. Zpracováno podle [16].

2.8.4 Prezentační vrstva

V nejjednodušším případě prezentační vrstva obsahuje určitá statická data, jako je statická HTML stránka. Typičtějším případem je dynamická stránka, která obsahuje data předaná z řadiče, která jsou v prezentační vrstvě jen zformátována požadovaným způsobem.

Ruby on Rails podporuje několik způsobů, jak generovat dynamický obsah. Nejčastěji užívanou šablonou je tzv. *ERb* (Embedded Ruby), která umožňuje do statického obsahu vložit kód v programovacím jazyce Ruby, jenž zajistí dynamický obsah. Tento nástroj je využíván ke generování dynamických HTML stránek a podporuje i pokročilejší způsoby vytváření výsledného dokumentu. Je tak možné, aby společné stránky sdílely stejné rozvržení umístěné v samostatné šabloně apod.

Jinou možností je tzv. *XML Builder*, jenž slouží k dynamickému generování XML dokumentů. Tento nástroj pro popis XML dokumentu využívá dynamických schopností jazyka Ruby a generuje výsledek na základě struktury kódu.

Další možností generování dynamického obsahu je šablona RJS, jenž umožňuje odeslat kód v JavaScriptu webovému prohlížeči, který jej vykoná. Této možnosti je využíváno v případech AJAXového volání. Podstatným rysem je, že změny, které se mají provést v klientském webovém prohlížeči, jsou popsány na straně serveru v jazyce Ruby a podle potřeby poté přeloženy do JavaScriptu. RJS tak poskytuje velice jednoduchý způsob vytváření AJAX aplikací, bez nutnosti zabývání se detaily. Funkčnost zabezpečují JavaScriptové knihovny Prototype a Scriptaculous. Vytvoření AJAXového volání pak spočívá v tom, že šablona RJS dá k dispozici objekt, reprezentující stránku v prohlížeči. Na tomto objektu existuje řada metod pro manipulaci s DOM modelem, vizuální efekty a další, jejichž volání je přeloženo na volání patřičných funkcí z výše zmíněných knihoven. Tímto způsobem je dosaženo transparentnosti a odpadá nutnost zabývat se implementačními detaily a nekompatibilitami mezi webovými prohlížeči. Zpracováno podle [16].

2.8.5 Řadiče

Řadič je centrem aplikační logiky celé aplikace. Jeho úlohou je spojení a ovládání všech komponent aplikace aby bylo dosaženo požadované funkčnosti. Řadiče v MVC modelu

koncentrují kód aplikační logiky na jednom místě, díky čemuž je pak aplikace daleko pružnější vůči budoucím změnám. Bázový řadič v Ruby on Rails, ze kterého dědí řadiče aplikací, již v sobě obsahuje mnoho funkčnosti, která zajistí základní požadavky na řadič webové aplikace.

Hlavním požadavkem na řadič v aplikaci, postavené na modelu MVC, je spojení událostí s konkrétními akcemi řadičů. V případě webových aplikací se jedná o spojení URL adres, na které mohou přijít síťové požadavky s akcemi řadičů.

K této činnosti je již v standardní distribuci frameworku Ruby on Rails k dispozici velice výkonný nástroj pro směrování, který je mimo jiné užitečný pro SEO optimalizaci. Tímto nástrojem je ve spojitosti s podporovanými regulárními výrazy možné provést prakticky libovolné mapování. Nicméně tato flexibilita je přístupná i v mnoha jiných, podobně zaměřených nástrojích. Ruby on Rails, navržený podle pravidla *Convention over Configuration*, usnadňuje práci s konfigurací směrování požadavků a přichází s výchozím modelem směrování, který je možné dále rozšiřovat.

Tento model vychází z architektury REST (Representational State Transfer), jejímž autorem je Roy Fielding, jeden z autorů specifikace síťového protokolu HTTP. Architektura je určena pro distribuované systémy, jakým je např. i WWW. Základním principem je skutečnost, že veškerá funkcionalita a stav vzdáleného systému je pro okolní svět reprezentován jako určitý zdroj/prostředek (angl. ressource). Každý takový zdroj je jednoznačně adresován a tato adresace podléhá určitým univerzálním syntaktickým pravidlům. Vedle samotné adresace je nutné, aby obě komunikující strany rozuměli obsahu komunikace. Z tohoto důvodu komunikující strany by měly být schopny se dohodnout na formátu komunikace, přičemž zvolený formát by se neměl dotknout samotných informací. Použitý síťový protokol by měl být typu klient/server, jelikož samotná architektura je založena na existenci uzlů, které distribuují svoje zdroje. Při komunikaci nejsou udržovány informace o stavu, čímž se redukují nároky na implementaci, ale i následný provoz systému, který je tak jednodušeji škálovatelný, např. pro použití prostředků ke snížení zátěže.

Principů REST architektury lze s výhodou využít v případě WWW technologií, které jsou dnes široce podporovány. Síťový protokol HTTP poskytuje způsob, kterým je

možné jednotným způsobem adresovat a přistupovat ke zdrojům. Jmenovitě se jedná o určení zdroje pomocí URI adresy, požadovaná operace se zdrojem je určena prostřednictvím HTTP metody, o výsledku operace nebo dotazu lze informovat pomocí návratových stavových kódů a dohodnutí o způsobu předání informace lze provést specifikací MIME typu.

Tyto principy lze uplatnit nejen v případě webových služeb, ale i v případě klasické webové aplikace. Za kandidáty na zdroje v RESTful systému budou v prvé řadě datové modely. Nad datovými modely je totiž nejčastěji potřeba provádět CRUD operace (vytvoření, čtení, aktualizace, odstranění). Nicméně jedná se pouze o rozhraní, takže zde není žádné omezení, které by vyžadovalo za zdroje považovat pouze datové modely aplikace. Jako příklad je níže v tabulce zobrazeno, jakým způsobem by mohla být řešena správa uživatelů ve webové aplikaci navržené podle REST principů a implementována v Ruby on Rails.

Operace	URL adresa	HTTP metoda	Akce v řadiči UsersController	URL webové služby s využitím XML
Přidání uživatele	/users	POST	create	/users.xml
Zobrazení všech uživatelů	/users	GET	index	/users.xml
Zobrazení detailu uživatele s id = 1	/users/1	GET	show	/users/1.xml
Editace uživatele s id = 1	/users/1	PUT	update	/users/1.xml
Odstranění uživatele s id = 1	/users/1	DELETE	destroy	/users/1.xml

Tabulka 1- Příklad REST adresace

Je konvencí, že jako jedinečný identifikátor zdroje je použit název zdroje v množném čísle. Jak z tabulky výše vyplývá, samotná URL adresa požadavku neurčuje požadovanou operaci se zdrojem, ta je určena až ve spojitosti s HTTP metodou. Pro operace, které mají za následek určitou změnu na serveru je vždy použita metoda jiná, nežli metoda GET, což je slučitelné s obecně uznávanými způsoby tvorby webů. Bohužel metody PUT a DELETE nejsou běžně implementovány ve webových prohlížečích a jiném softwaru, proto jsou tyto metody často nahrazeny metodou POST, přičemž se využije určitých mechanismů, které tyto metody simulují. V předposledním sloupci jsou uvedeny implicitní názvy akcí příslušného řadiče, které jsou implementovány jako metody třídy UsersController. Tento řadič bude vybrán na

základě výchozího směrování, jelikož jeho název tvoří požadovaná URL adresa následovaná řetězcem „Controller“, mimo další, pro tuto chvíli nevýznamné implementační podmínky. V posledním sloupci jsou uvedeny URL adresy pro webové služby využívající pro komunikaci formát XML. Je vhodné si všimnout, že tyto webové služby jsou dostupné pomocí stejných HTTP metod a implementovány ve stejných akcích. Jediný rozdíl spočívá v určení formátu na konci URL adresy, který je v případě absence nastaven na HTML. Podle zvoleného formátu je řadičem určen pohled, který je použit ke generování odpovědi. V každém případě je tímto způsobem dosaženo univerzální adresace zdrojů, určení výměnného formátu a dalších požadavků, kladených na RESTful systémy. Zpracováno podle [16].

2.9 Rich Internet Applications

Původním záměrem webu při jeho vzniku byla jednoduchá distribuce dokumentů, uložených na centrálním serveru. V průběhu doby se začaly objevovat servery, jejichž stránky byly dynamicky generovány určitými skripty, což byly první webové aplikace. Tento koncept se velice rychle rozšířil díky jednoduchosti centrálního nasazení a dostupnosti aplikace. Oproti desktopovým aplikacím totiž odpadají problémy s přenositelností mezi operačními systémy a aplikace je vždy distribuována v poslední dostupné verzi na serveru, čímž není potřeba se zabývat zpětnou kompatibilitou. Slabinou webových aplikací je ale jejich použitelnost, ve které dominují desktopové aplikace, využívající všech možností provozované platformy.

Určitými milníky v použitelnosti webových aplikací byla podpora programovacího jazyka JavaScript ve webových prohlížečích a pozdější podpora AJAX technologie, která umožnila na pozadí komunikovat se serverem a aktualizovat tak pouze potřebné části stránky, bez nutnosti jejího celého znovunačtení. Pro takové aplikace, splňující určité charakteristiky desktopových aplikací, vznikl termín *Rich Internet Application*, známější spíše pod svojí zkratkou *RIA*. Předpokládalo se, že tyto nové možnosti přenesou webové aplikace do popředí a v konečném důsledku vytlačí ty desktopové. To se ovšem nestalo. Jádro problému spočívá ve skutečnosti, že web byl zamýšlen jako systém pro distribuci jednoduchých dokumentů. Jakmile vznikaly snahy o přesunutí co možná největší části aplikační logiky na klientskou stranu, tím se začaly objevovat

problémy s kompatibilitou mezi prohlížeči, a kompatibilita byla paradoxně jeden z hlavních důvodů opuštění desktopových aplikací ve prospěch webu.

V současné době existují funkční řešení RIA aplikací, využívající možnosti AJAXu. V této oblasti vznikla velká iniciativa pro vytěžení maxima z doposud dostupných technologií. Druhou cestou k vytvoření RIA aplikace je využití určitého zásuvného modulu pro webový prohlížeč, který poskytuje běhové prostředí. V současnosti je k dispozici více takovýchto rozšíření od různých výrobců. Mezi nejrozšířenější patří Adobe Flash/Flex, Java a JavaFX od společnosti Sun Microsystems nebo Silverlight od společnosti Microsoft. Zpracováno podle [1].

Přestože je možné vytvořit kvalitní webovou aplikaci, založenou na technologii AJAX, považuji toto jen za obcházení nedostatků, jejichž příčina vychází z původního záměru webu a tím je distribuce dokumentů. V současnosti existují nástroje, které umožňují vývoj takových aplikací konvenčním přístupem, jako je Google Web Toolkit pro platformu Java, nebo JavaScriptová knihovna Dojo a mnoho dalších. Takové nástroje dokážou významným způsobem zvýšit použitelnost webových aplikací, přesto ale jejich možnosti nejsou na stejné úrovni, na které jsou RIA aplikace postavené na určitém zásuvném modulu. Proto se v následujícím textu zaměřím právě na ně.

2.9.1 Adobe Flex

Adobe Flex je soubor technologií určených k vývoji a distribuci RIA aplikací, běžících na platformě Adobe Flash. Samotný Flash je určen pro interaktivní bannery a animace, vložené do webových stránek. Tato platforma je široce rozšířená, průzkumy uvádí, že je dostupná přibližně na 98% desktopů. Značnou výhodou Flashe je jeho nezávislost na operačním systému, k dispozici jsou distribuce pro dnes všechny nejpopulárnější operační systémy.

Jádrem této platformy je objektově orientovaný programovací jazyk ActionScript, který vychází ze standardu ECMAScript. Aplikace na této platformě mohou využívat pokročilých funkcí pro práci s 2D, v poslední verzi 10 i s 3D grafikou, mohou využívat výhod více-vláknového zpracování a mají k dispozici velkou programovou knihovnu, podporující různé komunikační protokoly, multimediální formáty apod. Mimo standardně dodávanou programovou knihovnu, díky široké komunitě, která se vytvořila

kolem této platformy, je možné na internetu získat mnoho dalších knihoven distribuovaných pod otevřenými licencemi. Při vývoji pro Flash Player není nutné se uzavřít pouze do rámce této platformy. Tzv. External API umožňuje JavaScriptu běžícímu ve webovém prohlížeči komunikovat s Flashovou aplikací. Pro komunikaci v obráceném směru je k dispozici Flex-AJAX Bridge, také nazývaný jako FABridge, jež umožňuje integrovat technologii AJAX s Flash aplikací.

Všechny výše zmíněné vlastnosti jsou samozřejmě platné i pro aplikace vytvořené pomocí technologie Flex. Příčinou k jeho vzniku je skutečnost, že Flash je primárně zaměřený pro tvorbu animací, kde hlavní roli představuje časová posloupnost. Takový koncept je nevhodný k tvorbě uživatelského prostředí, které je spíše řízeno na základě vzniklých událostí. Flex tak poskytuje framework, který je vytvořený na platformě Flash a poskytuje prostředky, potřebné k tvorbě uživatelských rozhraní. Výsledné aplikace jsou zkompilovány do SWT souboru, ve kterém jsou obsaženy i Flexové knihovny, takže výsledný soubor lze distribuovat stejně jednoduše jako by se jednalo o čistý Flash.

Základem technologie Flex je volně dostupná sada nástrojů, označovaná jako *Standard Development Kit* – SDK, která obsahuje všechny potřebné knihovny a kompilátor. Pro vývoj aplikací se využívá dvou programovacích jazyků: MXML – *Magic eXtensible Markup Language* a ActionScript. MXML je jazyk založený na XML, jenž je vhodný k rozvržení ovládacích prvků aplikace, zatímco v ActionScriptu je popsána logika, řídící běh celé aplikace. MXML neposkytuje žádnou dodatečnou funkčnost oproti ActionScriptu, je určen pouze k strukturovanému popsání rozvržení aplikace. Ve skutečnosti je při kompilaci převeden na ActionScript, který je následně zkompilován. Samotný běh aplikace je celý řízen pomocí událostí. Nejedná se pouze o uživatelské události, ale i události generované samotnou aplikací. Podpora událostí je již obsažena na úrovni jazyka ActionScript, což značně ulehčuje vývoj. Zpracováno podle [1].

2.10 SEO

Search Engine Optimization, spíše známý pod termínem SEO, se zabývá technikami, jak dosáhnout ve výsledcích vyhledávačů při hledání konkrétních klíčových slov co nejvyšších pozic. Drtivá většina návštěvníků přichází na webové stránky z některého vyhledávače, který po zadání určitého dotazu usoudil, že ten který web je pro danou

oblast zájmu relevantní. Výsledky vyhledávání jsou ovšem dlouhé a ze statistik vyplývá, že uživatelé téměř v 90% případů hledají relevantní webové stránky na první stránce výsledků, dál jak na třetí stránku prochází jen nepatrný zlomek uživatelů. Je tedy potřeba se ve výsledcích vyhledávání dostat co nejvýše a toho je možné dosáhnout prostřednictvím tzv. on-page a off-page faktorů, které posuzují vyhledávače a podle nich hodnotí celkovou významnost a relevantnost stránek.

Vyhledávače se v minulosti potýkaly s množstvím nepoctivých praktik, které měly ovlivnit schopnosti vyhledávačů racionálně zhodnotit obsah a význam webových sídel, za účelem dosáhnout vyššího hodnocení. To zapříčinilo, že vyhledávače dnes velice detailně zkoumají, zdali se nejedná právě o některou již známou podvodnou praktiku. V kladném případě vyhledávač takové stránky záměrně penalizuje a to i naprostým vyřazením z indexu. V konečném důsledku to znamená, že maximalistický přístup k SEO optimalizaci může vést k diametrálně opačným výsledkům. Zpracováno podle [8].

2.10.1 On-page faktory

Mezi on-page faktory při optimalizaci webu pro vyhledávače spadají všechny vlastnosti a prvky, vyskytující se na hodnocené stránce. Podvodné praktiky některých provozovatelů webů zapříčinily, že tyto faktory jsou v poslední době méně zohledňované, při výpočtech hodnocení stránek vyhledávači. Přesto mají stále svoje nezastupitelné místo v SEO optimalizaci. Hlavní místo zastává především správná sémantika s vhodně zvolenými klíčovými slovy. Jestliže je vyhledávač schopen určit, co je hlavním informačním posláním zkoumané stránky, pak je taková stránka vhodně zařazena k danému tematickému okruhu v databázi vyhledávače.

Za nejdůležitější zdroj informací o obsahu stránky je považován její titulek. Vedle dalších značek, které lze využít k popisu metainformací o stránce, značka TITLE zůstává jediná, které je skutečně vyhledávači přiřazen podstatný význam. Metaznačka KEYWORDS je dnes považována za přežitek, jelikož se předpokládá, že její obsah vyhledávače již vůbec neanalyzují. Nicméně v případě metaznačky DESCRIPTION lze vypozorovat, že např. Google v zobrazení výsledků vyhledávání, v některých případech použije obsah této metaznačky v popisu odkazu. Z toho lze usuzovat, že je stále vyhledávači analyzována navzdory některým opačným názorům.

Dalším významným prostředkem při SEO optimalizaci, je použití tzv. user-friendly adres. V takovýchto adresách se vyskytují klíčová slova, která vypovídají o obsahu stránky. Vyhledávače analyzují URL adresy a klíčovým slovům v nich obsažených přikládají velkou váhu. Důležitou skutečností je také fakt, že stránky se staticky vypadající adresou jsou zvýhodněné, před adresami obsahujícími parametry apod., které mohou být vyhodnoceny jako adresy s výsledky vyhledávání na daném webu. Zpracováno podle [8].

2.10.2 Off-page faktory

Do těchto faktorů jsou zařazováni externí činitelé, kteří ovlivňují relevantnost k danému tématu a popularitu stránky. V zásadě se jedná o budování zpětných odkazů. Vyhledávače pomocí zpětných odkazů kategorizují weby podle zaměření odkazujících se webů. Kategorizace založená pouze na tomto mechanismu by ovšem nebyla zcela funkční vzhledem k cíleným výměnám odkazů mezi spřátelenými weby. Vhodným výběrem webů s podobným zaměřením ovšem lze dosáhnout dobrých výsledků, zvláště pak, když vyhledávač nabude dojmu, že se jedná o určitou autoritu v dané oblasti, na kterou se odkazují obdobně zaměřené weby.

Optimalizace, jejímž hlavím cílem je maximální počet zpětných odkazů, nemusí fungovat. Vyhledávače se snaží odhalit podvodné, uměle vytvořené zpětné odkazy, které nemají žádnou informační hodnotu. Nejlepším řešením v dlouhodobém horizontu proto bývá, vytvořit takový web, který bude mít pro návštěvníky určitou hodnotu a přínos. Slabinou u většiny firemních prezentací je skutečnost, že kromě kontaktních informací a pár odstavců o historii firmy, které nikdo nečte, pro návštěvníka takový web má malou informační hodnotu. Jestliže se ovšem firma rozhodne např. publikovat novinky z oboru, její prezentace získá nový rozměr. Mimo možné zvýšení počtu zpětných odkazů se na webu zvýší výskyt klíčových slov z oboru podnikání, což patřičně ohodnotí i vyhledávače, kteří začnou nabízet takový web ve výsledcích na tato klíčová slova. Pravidelné publikování novinek samozřejmě s sebou nese i svoje náklady, které vyvolávají neochotu k tomuto přístupu. Důležité ale je si uvědomit, že tyto náklady mají potenciál na své zhodnocení.

3 Analýza problému a současné situace

Tato část práce se zaměřuje na současné situace na poli webových prezentací a jejich redakčních systémů. Jejím účelem je prozkoumat současná řešení a poskytnout vstupní analýzu požadavků na vytvářený redakční systém.

3.1 Systémy pro správu obsahu

V současnosti existuje doslova nespočet systémů, určených ke správě obsahu webových sídel. Tyto systémy se nejčastěji rozdělují do dvou kategorií podle dostupné licence, upravující možnosti jejích užívání. Jmenovitě se jedná o kategorie:

- Komerční software
- Svobodný software

Komerční systémy pro správu obsahu jsou poskytovány za úplatu. Jejich cena je obvykle poměrně vysoká. Naproti tomu, systémy distribuované jako svobodný software jsou poskytovány zdarma, nejčastěji jako open source projekty, tedy i se zdrojovým kódem. Systémů tohoto typu existuje velké množství a lze mezi nimi nalézt jak velmi jednoduchá řešení, tak i komplexní, univerzálně navržené systémy. Důvodem stálé existence komerčních systémů na trhu je fakt, že zprovoznění volně dostupných systémů je často komplikované. U komerčních systémů jejich prodejce v zásadě poskytuje technickou podporu, popřípadě zajišťuje komplexní správu až po tvorbu nové grafické podoby webu. Problémy spojené s volně šiřitelnými systémy tedy ani tak nespočívají v tom, že by byly obtížněji konfigurovatelné, ale spíše v tom, že je k jejich instalaci a provozu potřeba určitých zkušeností a znalostí daného systému.

Dalším aspektem, při volbě mezi komerčním nebo svobodným softwarem, je jeho schopnost a náklady spojené s rozšířením funkčnosti o přídavné moduly. Komerční systémy jsou zpravidla nabízeny v několika variantách, které se od sebe liší v podporované funkčnosti. Jednotlivé varianty jsou samozřejmě z podnikatelských důvodů nastaveny tak, aby poskytovaly jen ty nejzákladnější funkce, přičemž další funkčnost je zpoplatněna. V těchto variantách jsou také obsaženy uměle vytvořené limity, jejichž překročení s sebou nese navýšení ceny. V případě svobodného softwaru toto samozřejmě neplatí. Přídavné moduly jsou poskytované zdarma a případné limity mají svoje technické pozadí.

V každém případě je milné se domnívat, že využití svobodného softwaru znamená, náklady blížící se nule. Padne-li volba na hotové softwarové řešení, pak při porovnávání komerčních a volně dostupných systémů je vhodné cenu za komerční řešení srovnat s náklady na instalaci a správu svobodného softwaru.

Kategorizace systémů pro správu webového obsahu lze provádět i podle mnoha jiných kritérií. Často zkoumaným rysem je použité vývojové prostředí, v němž je systém vytvořen. Potřebné technologie k provozu daného systému mohou významně ovlivnit náklady na provoz, ale samozřejmě také kvalitu. V případě svobodného softwaru většina CMS systémů využívá PHP ve spoluprácí s databází MySQL, což je výhodné vzhledem k široké podpoře těchto technologií u poskytovatelů hostingů.

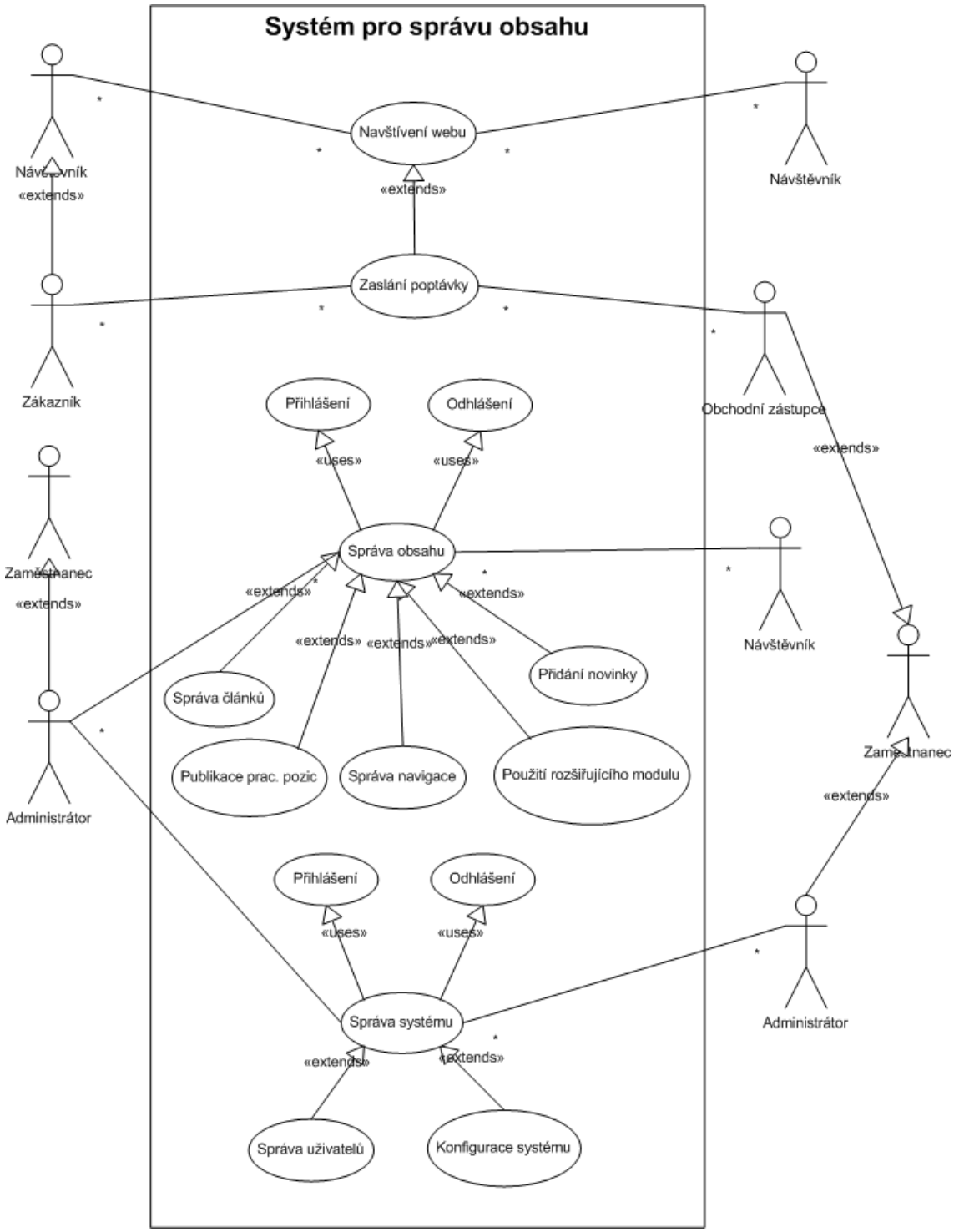
Důležitým kritériem je jistě také specializace systému. K dispozici jsou řešení ke konkrétním účelům, jako jsou blogy, diskusní fóra, foto galerie, redakční systémy apod. Vedle specializovaných systémů lze najít více či méně univerzální systémy, schopné naplit různorodé požadavky. Univerzálnosti je ovšem dosaženo na úkor jednoduchosti a tak tyto systémy většinou nejsou vhodné pro technicky méně znalé uživatele.

3.2 Analýza navrhovaného systému

3.2.1 Případy užití

Analýza případů užití – Use cases – poskytuje dynamický pohled na vytvářený systém. Hlavním smyslem této analýzy je zachytit systém z pohledu uživatele tak, jak jej bude využívat. Důraz je přitom kladen na vysokoúrovňový úhel pohledu, nezabývající se implementačními detaily.

Obrázek zachycující model případů užití ukazuje hlavní scénáře a jejich účastníky, jejž mohou nastat v průběhu provozu systému. Z této vstupní analýzy se bude následně vycházet při návrhu a realizaci výsledného systému.



Obrázek 2 - Model případů užití

3.2.2 Funkční požadavky

Hlavním účelem redakčního systému je správa obsahu, což je tedy i stěžejním funkčním požadavkem na vytvářený systém. Význam pojmu „obsah“ je ovšem široký a lze jej

minimálně rozdělit na obsah strukturovaný a obsah nestrukturovaný. Z tohoto rozdělení se bude vycházet při navrhování systému.

Pro strukturovaný obsah, který se skládá z více samostatných údajů, je vhodné tyto údaje uchovávat ve strukturované podobě, což v budoucnu umožní práci na úrovni jednotlivých údajů. Díky tomu strukturovaný obsah je možné na základě různých kritérií jednoduše kategorizovat, řadit, prohledávat ho apod. Oproti tomu nestrukturovaný obsah je směsice neuspořádaných informací, které lze strojově zpracovávat jen velice obtížně. Jeho největší předností je univerzálnost, která dovolí v oblasti webu alespoň vizuální tvorbu libovolného obsahu.

Pro univerzálnost by výsledný systém měl podporovat práci s nestrukturovaným obsahem. Na takový obsah se bude odkazovat jako na tzv. článek, který obsahuje formátovaný text ve spojitosti s možností vkládání obrazových informací v podobě obrázků. Takovýto článek pak představuje celou WWW stránku.

Správa strukturovaného obsahu vyžaduje tvorbu samostatných dílčích částí určitých modulů. Ty jsou navrženy podle struktury obsahu a je požadováno s nimi specificky zacházet. I tento druh obsahu ovšem v sobě může, a ve většině případů také bude, obsahovat nestrukturované údaje z důvodu uživatelské přívětivosti a variabilitě aplikace. V navrhovaném systému jsou požadovány funkce ke správě poptávek, novinek, navigace webu, volných pracovních pozic a kontaktních informací.

Jak je ilustrováno na modelu případů užití, přímé účastníky systému lze rozdělit do dvou obecných rolí a to na návštěvníky webu a zaměstnance. Zaměstnanci, kteří se starají o obsah webu, by měli mít přístup k administrační části. To samozřejmě neplatí o návštěvnících, jimž by měl být povolen pouze přístup do veřejných částí webu. Tento požadavek si vynucuje tvorbu uživatelského podsystému, jenž zabezpečí autentizaci uživatelů a v případě potřeby zamezí nedovolenému použití aplikace. Autorizační pravidla, upřesňující jaké konkrétní operace, smí daný uživatel provádět, v tomto případě ale nejsou zapotřebí. To plyne z požadavků zadavatele, jenž bude výsledný systém využívat v minimálním počtu zaměstnanců a tak by tato funkčnost byla spíše překážkou než přínosem.

Důležitým aspektem je, aby systém byl otevřený vůči případným rozšiřujícím modulům. Tyto moduly by přitom především spadaly mezi funkčnosti pro správu publikovaného obsahu, proto je v modelu případů užití tento abstraktní případ užití odvozen od případu užití „Správa obsahu“.

3.2.3 Další požadavky

Navrhovaný systém je určen pro řadového uživatele. Je tedy zapotřebí, aby jeho uživatelské rozhraní bylo snadno pochopitelné a nevyžadovalo určité odborné znalosti webových technologií k jeho ovládání. Samozřejmostí pro moderní web je graficky působivé zpracování publikovaného obsahu.

Systém by dále měl být odolný vůči bezpečnostním útokům. Tento požadavek bohužel nelze zcela naplnit nikdy, ovšem se znalostí populárních bezpečnostních útoků lze vybudovat dostatečně odolný systém. Drtivá většina útoků totiž vychází z již známých bezpečnostních mezer.

4 Vlastní návrhy řešení

Cílem této části je navrhnout systém pro správu obsahu. Smyslem tvorby tohoto systému je umožnit spravovat obsah webové prezentace i uživatelům, bez znalostí webových a s nimi spojených softwarových technologií.

Mimo samotný systém s uživatelsky přístupným rozhraním se tato část zaměří na oblast samotného budování webových aplikací. Je jisté, že pokud má být systém jednoduchý na ovládání, musí být navržen ke konkrétnímu účelu. Zvyšování univerzálnosti koreluje se složitostí konfigurace a správy systému. Možným řešením je vytvoření základního systému, do kterého budou přidány moduly vytvořené na klíč ke konkrétnímu účelu. Součástí této práce tedy je i poskytnout vhodný technologický rámec pro rychlé a efektivní vytváření rozšiřujících částí systému.

4.1 Výběr technologií

Klíčovým rozhodnutím na počátku realizace softwarového projektu je výběr nástrojů a technologií, pomocí kterých bude provedena realizace. Tato rozhodnutí velkým způsobem předurčí parametry výsledného systému. V současné době na trhu existuje velké množství nástrojů určených k tvorbě webu. V zásadě se jedná o komplexní nástroje, a proto jejich srovnání není jednoduché. Při výběru je potřeba vycházet z konkrétních požadavků plynoucí ze zadání projektu, které již možnosti volby významně zúží.

4.1.1 Aplikační vrstva

V současné době je nejpopulárnějším prostředím k tvorbě dynamických webů PHP, které je také podporované snad všemi poskytovateli webhostingů. Použití PHP v sobě ovšem skrývá množství nevýhod. Tento jazyk vychází z jednoduchého procedurálního přístupu, a přestože určitou podporu OOP obsahuje, jeho vestavěné funkce ji nevyužívají, což znemožňuje objektový přístup nad takovýmto jádrem. Toto prostředí nepřináší unifikovaná řešení pro problémy, které jsou nedílnou částí již trošku komplexnějších aplikací. Namísto toho řešení je ponecháno na programátorovi, což ovšem neplatí u jiných, propracovanějších technologií.

Za populární technologie k tvorbě webových aplikací lze považovat také ASP.NET a J2EE. Nicméně jejich popularita, co do počtu běžících webových aplikací, nebo

poskytovatelů webhostingů, nedosahuje takové úrovně jako je tomu u PHP. Na druhé straně se jedná o plnohodnotné nástroje, pomocí kterých je možné vytvářet i velice komplexní systémy. Tato navzájem konkurenční prostředí jsou určená zejména pro tvorbu složitějších aplikací, proto jejich použití v případě menších systémů bývá těžkopádné.

Relativně novou technologií je framework Ruby on Rails. V současnosti jeho podpora mezi českými webhostingy není velká, na druhé straně, toto prostředí umožňuje vytvářet webové aplikace způsobem podobným pro komplexní systémy, přičemž netrpí problémy s těžkopádností. Velice zhruba jej lze zařadit někam mezi dvě dříve zmiňované kategorie. Použití Ruby on Rails není tak jednoduché bez znalostí o určitých stavebních principech, oproti zmiňovanému PHP. Na druhou stranu není tak těžkopádné a deklarativní jako je tomu u ASP.NET a J2EE, ale přesto umožňuje tvorbu čistého a znovupoužitelného kódu ve spojitosti s osvědčenými návrhovými vzory. Z těchto důvodů považuji za vhodné, navrhovaný systém o této velikosti implementovat pomocí Ruby on Rails.

4.1.2 Databáze

Nedílnou součástí systému pro správu obsahu je datové úložiště. K tomuto účelu zcela vhodně poslouží některá relační databáze. Zcela jistě dnes nejpodporovanější databází mezi poskytovateli webhostingů je MySQL. Jak již bylo zmíněno v teoretické části této práce, MySQL nedisponuje celou řadou vlastností, které jsou jinak běžné u jiných databází. Nicméně při využití MySQL ve verzi 5 bude tato databáze zcela postačující, vzhledem k propracované podpoře práce s relačními databázemi v Ruby on Rails. Náročnější funkčnosti tak budou implementovány na úrovni datového modelu aplikační vrstvy.

4.1.3 Prezentační vrstva

Navrhovaný systém se skládá ze dvou uživatelských prostředí, které by se daly označit jako veřejný prostor pro publikaci obsahu a administrátorské rozhraní, určené ke správě prezentace. Tato prostředí jsou postaveny na odlišných technologiích, z nichž každé má svoje výhody k danému účelu.

Podsystém určený k prezentaci obsahu je vystaven na běžných webových technologiích, jejichž základem je HTML dokument. Tím je zajištěna vysoká přístupnost návštěvníků k obsahu webu, jelikož k jeho zobrazení není zapotřebí, aby na klientské straně byly podporovány některé pokročilejší technologie. Neméně významným důvodem je skutečnost, že internetové vyhledávače primárně pracují s HTML dokumenty, které jsou schopny patřičně zanalyzovat. Stránky, které nevycházejí z klasického modelu všeobsáhlých informací v rámci dokumentu, ale namísto toho vyžadují podporu určitých pokročilejších technologií, jako je JavaScript, AJAX, zásuvné moduly apod., pak vyhledávače nemohou adekvátně prozkoumat a v konečném důsledku jim tak přiřadí nižší hodnocení. Tím v žádném případě nechci zpochybňovat užitečnost JavaScriptu a AJAXu, ale jen zdůraznit nutnost alternativního obsahu, dostupného i v případě absence těchto technologií.

Vedle toho, pro častou práci s administrátorským rozhraním, webové prostředí má relativně dlouhou odezvu a nepodporuje takové možnosti, jako desktopové aplikace. Pro odstranění těchto nedostatků lze s výhodou použít běhové prostředí Adobe Flash Player, které je dostupné a bezproblémově přenositelné mezi všemi populárními operačními systémy. Jak již bylo uvedeno v teoretické části této práce, pro zmíněné prostředí je k dispozici Adobe Flex, určený k tvorbě RIA aplikací. Pomocí této technologie je vytvořena administrátorská aplikace pro přístup k systému.

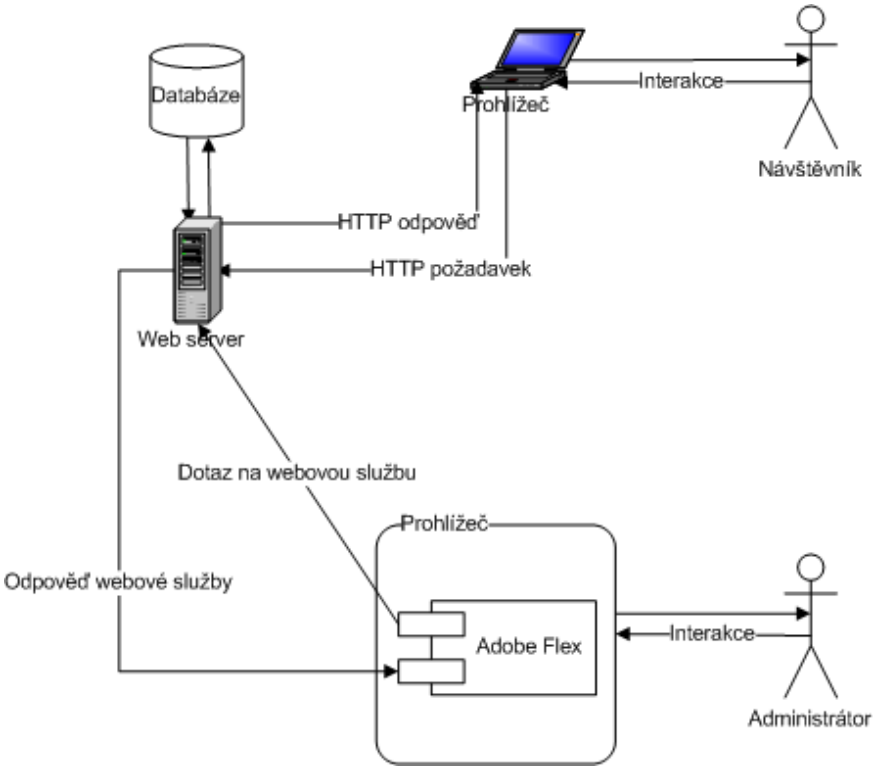
Na webovém serveru je umístěna tato aplikace v souborovém formátu SWF. To umožňuje snadnou aktualizaci verzí, pouhým přepsáním souboru, oproti případu desktopové aplikace, instalované na lokálním systému.

Administrátor si stáhne tuto aplikaci skrze webový prohlížeč, v němž přejde na patřičnou adresu HTML stránky, ve které je umístěna tato aplikace. Aplikace se poté spustí v rámci prohlížeče a celá její činnost se odehrává na straně klienta, což umožňuje na klientskou část přenést více aplikační logiky systému a tak zvýšit jeho odezvu. V případě větších systémů tak lze samozřejmě i docílit nižšího zatížení centrálního serveru.

4.2 Architektura navrhovaného systému

Přístupem zmiňovaným v předchozí části, vzniká nutnost vytvoření určitého komunikačního rozhraní mezi klientskou aplikací a aplikačním serverem. V neposlední řadě, při nevhodně navržené synchronizaci dat mezi serverem a klientem, může být těžké udržovat data v konzistentním stavu tak, aby byla na obou stranách opravdu totožná. Vhodné řešení poskytuje architektura REST, jenž definuje jednotnou adresaci zdrojů na serveru a přitom nevyžaduje udržování stavu mezi jednotlivými komunikacemi. To významně ulehčuje implementaci takového distribuovaného systému.

Klientská Flex aplikace tak přímo komunikuje s aplikačním serverem, jenž požadovaná data získává z databáze. Tato data jsou následně přetransformována do klientem požadovaného formátu a výsledek je odeslán zpět klientovi. V konečné fázi RIA aplikace zobrazí doručená data uživateli. Vysokoúrovňový pohled na popisovanou architekturu navrhovaného systému ilustruje následující diagram.



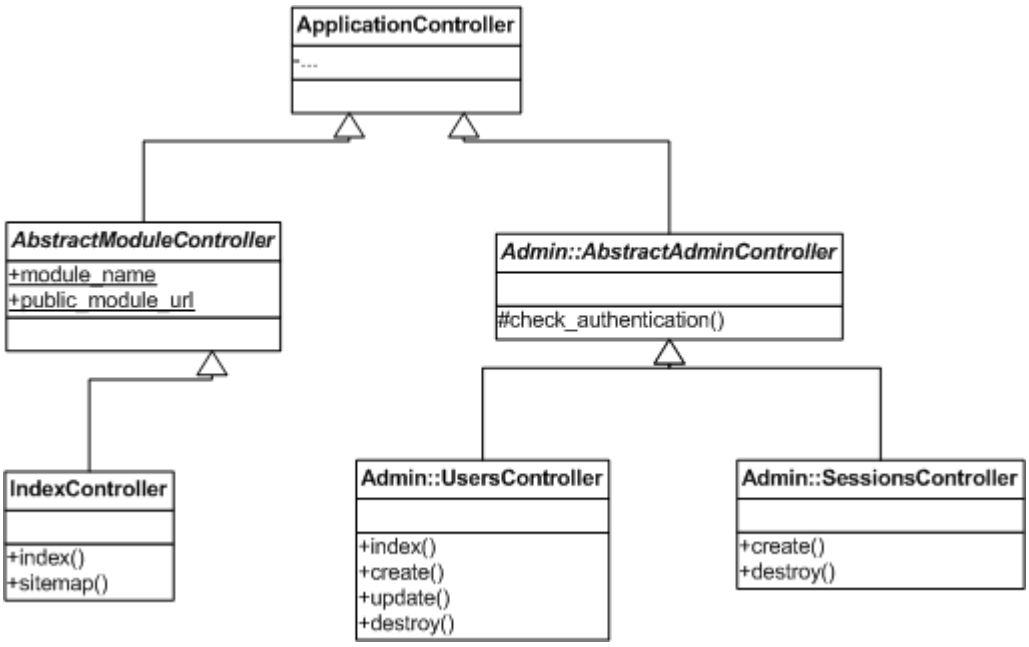
Obrázek 3 – Základní architektura systému

4.3 Funkční model

Základem funkčního modelu v aplikaci navržené pomocí architektury MVC jsou řadiče, které v sobě obsahují tzv. akce. V akcích je potom popsána aplikační logika systému, vycházející z definovaných požadavků.

Navrhovaný systém je postaven na principech REST, tedy na logické úrovni systému jsou umístěny zdroje, reprezentující jak entity, tak i stavy systému. Základními operacemi nad těmito zdroji jsou tzv. CRUD – Create Read Update Delete – akce, které jsou dostupné pro většinu zdrojů. U některých zdrojů se jedná pouze o podmnožinu těchto operací, popřípadě podporovaných operací může být více. Výhodou tohoto přístupu k základním operacím se zdroji je jednotné mapování síťových požadavků na konkrétní akce v aplikaci. Přijde-li požadavek používající HTTP metodu GET na kořenovou adresu zdroje, pak je konvenčním způsobem namapován na akci *index* příslušného řadiče. V případě metody POST se vykoná akce *create* atd. Způsob mapování byl již popsán v teoretické části této práce, viz. Tabulka 1- Příklad REST adresace. Důležitou skutečností je rozdělení komplexního systému na dílčí části s jednoduchým rozhraním. Výhodou tohoto přístupu je přehledná orientace v systému a jeho snadné rozšíření v budoucnosti o přídavné moduly.

Společnou bázovou třídou pro všechny řadiče v systému je *ApplicationController*. Ten víceméně jen zabezpečuje základní funkce potřebné k chodu systému. Na základě specifikace požadavků jsou řadiče dále rozděleny do dvou skupin podle toho, zdali se jedná o řadič administrátorského prostředí, nebo o řadič určený k veřejné publikaci obsahu. Pro přístup k administrátorským řadičům je systémem požadována autentizace uživatele na základě uživatelského jména a hesla. Podpora této funkčnosti je umístěna v abstraktní třídě *AbstractAdminController* umístěné v modulu *Admin* (stejně jako zbytek administrátorských řadičů). V této bázové třídě je umístěn tzv. *before_filter*, jenž je vykonán před spuštěním jakékoliv akce v něm, nebo v odvozených třídách. Tento filtr tak před obsluhou každého požadavku provede ověření, zdali je operace podnícena autentizovaným uživatelem a v případě záporného zjištění zamítne provedení požadované akce.



Obrázek 4 – Část statického podhledu na funkční model systému

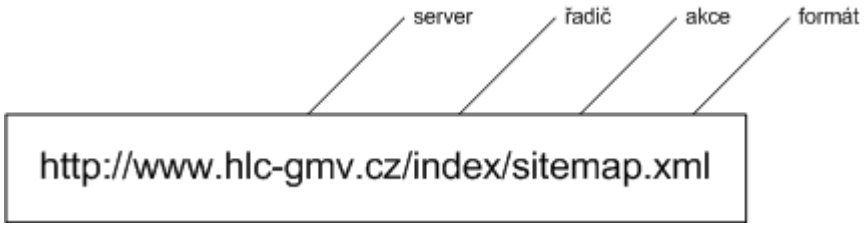
Společné chování pro řadiče určené k veřejné publikaci obsahu je definováno v abstraktní třídě *AbstractModuleController*. Pro tyto řadiče lze také za rodičovskou třídu použít přímo *ApplicationController*, nicméně hlavním důvodem pro existenci třídy *AbstractModuleController* je rozdělení veřejně dostupných řadičů do skupin pro moduly systému a podpůrné řadiče těchto modulů. Aplikace je tak schopna sama osobě určit dostupné nainstalované moduly v systému, bez nutnosti programové nebo ruční údržby určitých seznamů. Této schopnosti bude využito v administrační části, vyhrazené ke správě navigačních nabídek webu.

4.3.1 Adresace zdrojů v systému

Pro příklad fungování celého systému jsou na Obrázek 4 – Část statického podhledu na funkční model systému, vyobrazené i určité konkrétní řadiče, poskytující již určitou funkcionalitu systému.

Řadič *IndexController* je kořenovým řadičem celého systému. Je v něm obsažena jak „konvenční“ akce index, která je vyvolána v případě požadavku GET na titulní stránku celého webu, tak akce *sitemap*, která je určená ke generování mapy webu. Taková akce nepatří do výchozích operací se zdroji, to ale neznamená, že by se jednalo o neslučitelný návrh s REST principy. Tato akce je automaticky směrovačem nalezena na adrese */index/sitemap*, v tomto případě kořenového řadiče je také dostupná na adrese

/sitemap. Poslední volitelnou částí adresy je určení požadovaného formátu, pro který je jako výchozí hodnota použito *html*. V případě generování XML sitemap pro internetové vyhledávače, se bude vycházet ze stejných dat od dané akce, přičemž se jen použije pohled ke generování XML dokumentu. Obecný princip adresace v celém systému je ilustrován na následujícím Obrázek 5 - Určení řadiče, akce a výstupního formátu z požadované adresy.

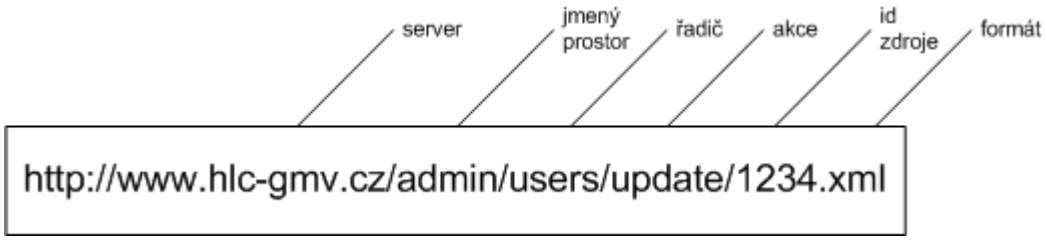


Obrázek 5 - Určení řadiče, akce a výstupního formátu z požadované adresy

Pro řadiče administračního rozhraní je vhodné poskytnout samostatný jmenný prostor. Toho je dosaženo umístěním řadičů do modulu[2] *Admin* a směrováním požadavků spadajících do administračního jmenného prostoru na tento modul.

Na ilustračním Obrázek 4 – Část statického podhledu na funkční model systému, jsou zobrazeny řadiče *UsersController* a *SessionController*. V případě prvního se jedná o realizaci základních operací pro manipulaci s určitým zdrojem umístěným v nějakém trvalém úložišti. Naproti tomu *SessionController* reprezentuje pouze stav aplikace – uživatelskou relaci. V tomto případě akce *create* zastupuje přihlášení uživatele do systému a akce *destroy* jeho odhlášení. Příklad adresace zdrojů v administračním jmenném prostoru je zobrazena na Obrázek 6 - Zpracování adresy v administračním prostředí. Zde je pro výsledný systém důležitá část specifikující formát odpovědi, který je důležitý pro klientskou aplikaci, komunikující s aplikačním serverem. Příklad ilustruje aktualizaci uživatelského profilu s identifikačním číslem 1234, přičemž na tuto adresu bude odeslán XML dokument obsahující nové údaje o uživateli.

[2] V tomto kontextu je modulem míněn prostředek *module* programovacího jazyka Ruby, pomocí kterého lze slučovat třídy a jiné programové předměty do jednoho organizačního celku. Tento význam nemá žádnou spojitost s moduly informačních systémů.



Obrázek 6 - Zpracování adresy v administračním prostředí

4.4 Datový model

Datový model za svoje úložiště využívá relační databázi. Databázové schéma z praktických důvodů zcela nesplňuje pravidla normálních forem. Aplikace k databázi nikdy nepřistupuje přímo, ale vždy použije datový model vytvořený pomocí ORM knihovny ActiveRecord. V datovém modelu jsou definována validační pravidla, která znemožní poškodit integritu databáze. ActiveRecord podporuje kromě validace i tvorbu tzv. callbacků, což jsou určité rutiny, které jsou automaticky vykonány v požadovaném okamžiku, jako je okamžik před nebo po: validaci, uložení, aktualizaci, vytvoření, odstranění záznamu apod. Callback může např. dopočítat hodnotu pro určitý sloupec před uložením záznamu do databáze, zajistit podporu aktivního integritního omezení apod.

Označení jednotlivých objektů databáze vychází z konvencí, které předpokládá knihovna ActiveRecord. Třídy z datové vrstvy aplikace hledají v databázi cílovou tabulku se stejným názvem, jako je název třídy, ale v množném čísle. Jako primární klíč je vždy použit sloupec *id*. ActiveRecord ve své základní verzi nepodporuje složené primární klíče, nicméně existuje rozšíření, zajišťující tuto funkcionalitu. Sloupce obsahující cizí klíč jsou pojmenovány po mateřské entitě v jednotném čísle, za nímž následuje řetězec „_id“. V namapované třídě jsou pak dostupné atributy se stejnými názvy, jako jsou názvy sloupců v namapované tabulce. Za povšimnutí ještě stojí sloupce s názvy *created_at* a *updated_at*, které představují časová razítka, kdy byl daný záznam vytvořen resp. aktualizován, přičemž hodnoty těchto sloupců jsou automaticky nastavovány zmíněnou ORM knihovnou.

Sloupce označené jako *permalink* jsou vytvořeny za účelem SEO optimalizace. Tyto sloupce jsou automaticky nastavovány, prostřednictvím *before_save* callbacku, na základě hodnoty ve sloupci, jenž představuje určitý titulek záznamu. Takovýto titulek je

převeden na malá písmena, jsou z něj odstraněny speciální znaky a diakritika a výsledná hodnota je pak použita v URL adrese za jedinečným identifikátorem daného záznamu.

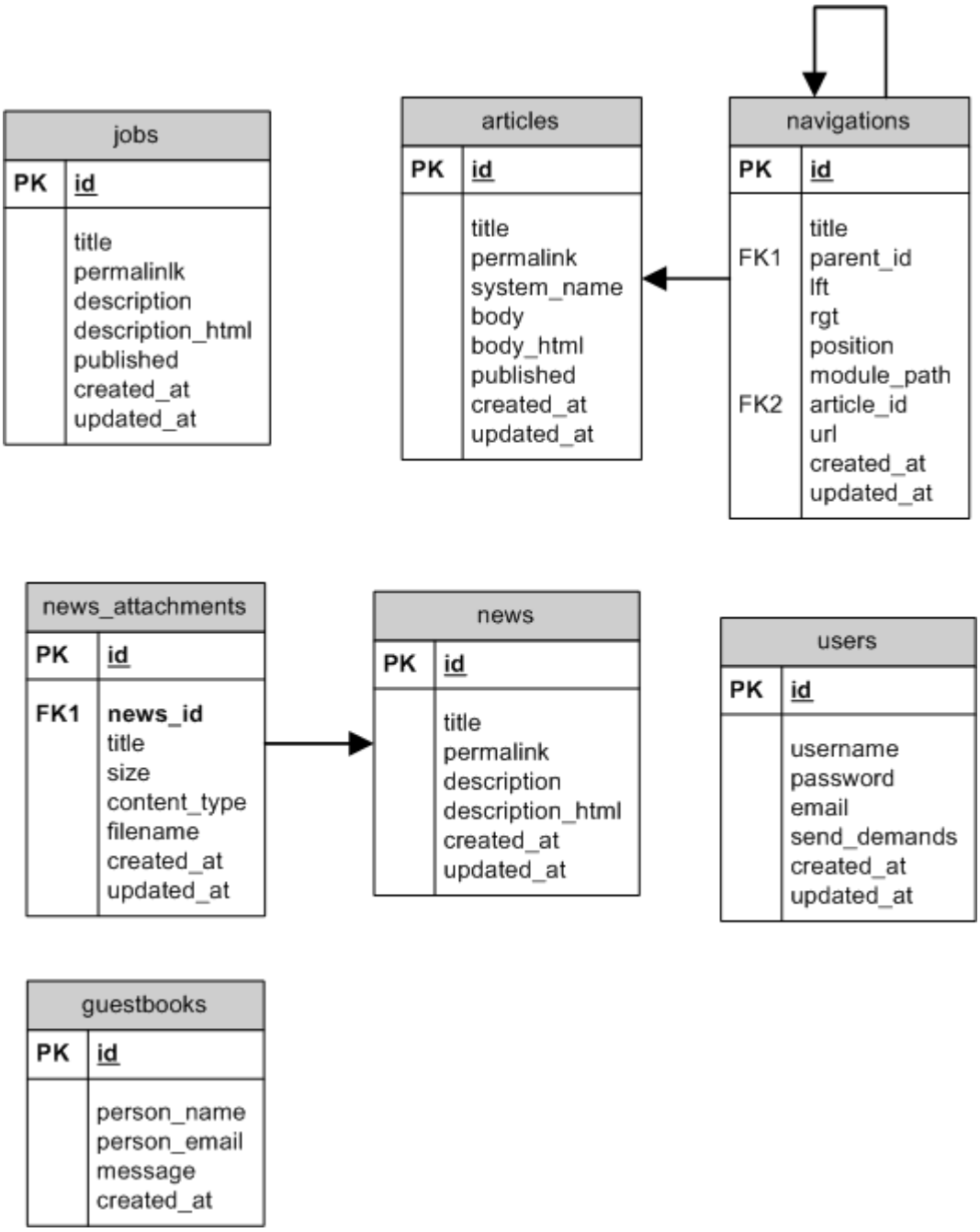
Sloupce, které se v tabulkách vyskytují ve dvou variantách a to sice s postfixem *_html* a bez něj, jsou určeny k uložení formátovaného textu, pocházejícího z WYSIWYG editoru v administračním rozhraní. Tyto sloupce jsou samozřejmě redundantní. Bylo by možné sloupce určené pro úschovu prostého textu simulovat v datové vrstvě odstraněním formátování. Nicméně formátování může být komplexnější a jednoduchým odstraněním HTML značek tak nemusí být výsledná podobna vhodná k použití, jako např. náhled obsahu. Proto tedy tato redundance představuje jistá zadní vrátka, která mohou být prospěšná v budoucnosti. V současné verzi aplikace je formátování odstraňováno programově.

Názvy sloupců jsou vesměs samo popisné až na pár výjimek. Tabulka *articles* obsahuje sloupec *system_name*, jenž slouží k určení specifického významu článku v rámci systému. V této základní verzi je jedinou přípustnou hodnotou *contacts*. Takovýto článek pak představuje stránku s kontaktními informacemi. Existence tohoto sloupce vychází z předpokladu, že pro provoz určitých modulů systému postačí podpora tvorby formátovaného textu, přičemž tento text má speciální účel. Aby nedocházelo k duplikaci kódu v administrační aplikaci a v publikační části redakčního systému, je vytvořen tento obecný přístup ke komponentě obsahující formátový text.

Tabulka *navigations* představuje strukturovanou nabídku dostupnou v publikační části webu. Položka nabídky, představovaná jedním záznamem v této tabulce, reprezentuje odkaz, který může směřovat buďto na absolutní URL adresu, určitý článek v redakčním systému, nebo titulní stránku určitého modulu systému. Podporu prvních dvou variant zabezpečují sloupce *url* a *article_id*. V případě odkazu na modul systému bude jeho relativní cesta uvedena ve sloupci *module_path*. Seznam nainstalovaných modulů a jejich relativních adres získá administrační aplikace skrze řadič *Admin::ModulesController*. Tento řadič využívá dynamických možností programovacího jazyka Ruby, pomocí kterých najde všechny dostupné implementace abstraktní třídy *AbstractModuleController* – tedy všechny nainstalované moduly. Z tohoto důvodu databázové schéma neobsahuje žádné informace o dostupných modulech a v tabulce *navigations* je použita pouze textová informace.

Stromová hierarchie nabídky je zabezpečena pomoci sloupců *parent_id*, *lft* a *rgt*. Jedná se o dva přístupy k udržování stromové struktury v relační databázi, z nichž každý má své výhody a nevýhody. Sloupec *parent_id* představuje cizí klíč na nadřazenou položku nabídky. Jedná se o nejjednodušší přístup, který je ale velice neefektivní při rekurzivním načítání celého stromu nebo jeho části po jednotlivých úrovních. Pro tyto případy lze s výhodou využít tzv. vnořených množin[3], jejichž funkčnost zabezpečí celočíselné sloupce *lft* a *rgt*.

[3] HILLYER, Mike. *Managing Hierarchical Data in MySQL*. [online] [cit. 2009-05-08] Dostupně z: http://dev.mysql.com/tech-resources/articles/hierarchical-data.html



Obrázek 7 - Databázové schéma

4.5 Administrační rozhraní

Administrační rozhraní je realizováno jako klientská aplikace běžící na platformě Flash, využívající technologie Flex, pro tvorbu aplikací. Vedle podpory pro tvoru RIA aplikací, běžících ve webovém prohlížeči pomocí zásuvného modulu Falsh Player, je možné aplikaci přizpůsobit pro provoz na platformě Adobe AIR. V tomto případě se jedná o aplikaci označovanou jako RDA – *Rich Desktop Application*, k jejímuž provozu

není zapotřebí webového prohlížeče, ale aplikace je nainstalována na lokálním systému, jako by se jednalo o klasickou desktopovou aplikaci.

Pro provoz administrační aplikace je nezbytná komunikace se serverem, na němž je umístěn systém pro správu obsahu. Tato aplikace tvoří prezentační vrstvu s uživatelským rozhraním pro ovládání systému. V tomto případě ovšem prezentační vrstva nepředstavuje hloupého klienta pouze pro zobrazování informací ze serveru. Flex aplikace mají přístup k širokému spektru pokročilých funkcí a tak prezentační vrstvou je spíše plnohodnotná aplikace, komunikující se serverem, na němž jsou uložená veškerá data systému.

4.5.1 Cairngorm

Přestože je možné Flex aplikace vytvářet bez pomoci jakýchkoliv dalších knihoven, mikroarchitektura Cairngorm, jak je označována ve své oficiální dokumentaci, zavádí do vývoje určité návrhové vzory, jejichž pomocí je možné zvýšit přehlednost a kvalitu vytvářeného systému. Jednoduché aplikace je možné rychle a snadno vytvořit bez nutnosti využití určitého návrhu, jenž zajistí strukturovaný kód. Tvorba větší aplikace bez kladení důrazu na strukturu kódu ovšem s největší pravděpodobností povede k chaotickému řízení a ve svém důsledku k chybám v implementaci.

Řízení Flex aplikací je založeno na generování a obsluze vzniklých událostí. I mikroarchitektura Cairngorm je ve své podstatě založena na tomto principu, přičemž poskytuje podporu MVC modelu při práci s těmito událostmi. Cairngorm se zaměřuje na aplikace, jenž představují prezentační vrstvu pro back-end systému, což je ve shodě s požadavky na vytvářenou aplikaci.

Knihovna se skládá z několika komponent, které mezi sebou komunikují, což vytváří chování systému. Některé komponenty ovšem pro navrhovanou aplikaci nejsou zcela vhodné, jmenovitě se jedná o část určenou pro přímou komunikaci s webovými službami, jenž není navržená pro REST architekturu, namísto toho vychází z konceptu vzdálených metod. To ovšem není překážkou v použití mikroarchitektury Cairngorm. Pro využití REST zdrojů na vzdáleném systému pro správu obsahu postačí jednoduchá funkce, jež naváže HTTP spojení na danou adresu zdroje a odešle požadovaná data ve formátu XML.

4.5.2 Implementace

Jak již bylo zmíněno, implementace administrační aplikace je postavena na mikroarchitektuře Cairngorm, která vychází z modelu MVC.

První částí systému je datový model, v němž je definována struktura dat, s nimiž manipuluje aplikace. Tento model je představován množinou tříd, které představují jednotlivé entity a jejichž atributy zastupují konkrétní údaje. Nedílným funkčním požadavkem na každý datový model je schopnost serializace údajů do podoby, kterou lze přenést po síti na vzdálený systém, kde budou zaslané údaje deserializovány s cílem následného zpracování. Pro takový komunikační protokol je použit formát XML, který je dobře podporovaný na obou stranách komunikace. XML dokumenty nepodléhají žádné DTD definici, ale jsou jednoduchým způsobem tvořeny vnější značkou s názvem modelu, v němž jsou obsaženy značky s názvy jednotlivých atributů modelu a jejich hodnoty. V případě přenosu více modelů v kolekci jsou všechny údaje ještě umístěny ve značce s názvem modelu v množném čísle, což zjednodušuje práci se skupinami objektů. Tento způsob je víceméně konvenční v případě frameworku Ruby on Rails, v případě platformy Flex je nutné podporu této funkčnosti zajistit v samotné aplikaci. Vzhledem k tomu, že ActionScript 3 obsahuje datový typ XML a funkce pro zpracování XML dokumentů již na úrovni jazyka, implementace v jednotlivých modelech je velice přímočará.

Důležitou součástí datového modelu je implementace rozhraní *com.adobe.cairngorm.model.ModelLocator* z mikroarchitektury Cairngorm, jenž zabezpečuje uchovávání veškerého stavu aplikace. V této komponentě systému jsou tak mimo jiné udržovány veškeré načtené modely ze serveru, ke kterým je možné dále přistupovat z různých částí aplikace. Tímto způsobem je tak zajištěno jednoduše reagovat na vzniklé události kdekoliv v systému.

Řadiče zastupují tzv. příkazy (angl. command), které představují jednotlivé operace v aplikaci. Tyto příkazy jsou implementovány ve spojitosti s obslužnými rutinami, které jsou spuštěny na základě úspěšnosti vykonávaného příkazu. Drtivá většina příkazů obsahuje určitou komunikaci se serverem. Ta je umístěna do samostatné vrstvy v podobě tzv. delegátů, díky čemuž je odstíněno komunikační rozhraní serveru od zbytku administrační aplikace, která tak na něm není závislá. Příkaz, který

komunikuje se serverem tak zavolá patřičnou metodu u daného delegáta, u nějž se zaregistruje k odběru událostí, spojených s komunikací v rámci dané operace. Delegát tak na základě úspěšnosti komunikace a odpovědi serveru zavolá patřičné metody příkazu, reprezentující úspěšnost, resp. selhání operace.

Prezentační vrstva aplikace představuje GUI komponenty, vytvořené výlučně na prvcích platformy Flex. Cairngorm tuto část plně přenechává na standardně dodávané funkčnosti ve Flex SDK balíku. Je důležité podotknout, že díky výše zmíněnému rozdělení aplikace, je prezentační vrstva očištěna od kódu, představující aplikační logiku. Tato vrstva se skládá z MXML komponent, v nichž jsou pomocí jazyka XML popsány struktury GUI komponent. Ty pak ze své podstaty generují uživatelské události, při jejímž výskytu jsou provedeny patřičné příkazy, což oživí celou aplikaci.

4.6 SEO optimalizace

Optimalizace pro internetové vyhledávače je dnes důležitou součástí moderních webových sídel. Vynaložené náklady se mohou velmi rychle vrátit jestliže se nejedná o některá vysoce konkurenční klíčová slova. V případě oboru podnikání zkoumané společnosti a to sice velkoobchodní prodej hydraulických výtahů, se jistě nejedná z pohledu internetových vyhledávačů o vysoce konkurenční prostředí. Přesto je nanejvýš vhodné realizovaný web optimalizovat pro vyhledávače, které budou minimálně schopné si spojit danou prezentaci s patřičnými klíčovými slovy. Pro získání vyšších pozic ve výsledcích vyhledávání jsou nutné zpětné odkazy, které již se samotnou technickou stránkou redakčního systému nijak nesouvisí.

Publikovaný obsah je založen na striktním oddělení obsahu o jeho vizuální podoby. Generované XHTML dokumenty představují samotný obsah, v němž je vhodným způsobem vyznačen sémantický význam jednotlivých částí. Grafická prezentace obsahu je zajištěna prostřednictvím kaskádových stylů, což mimo jiné zjednodušuje správu a případnou změnu vizuální podoby webu.

Použité adresy v rámci webu jsou tzv. user-friendly. Takové adresy neobsahují nic neříkající směsici parametrů, ale připomínají statické adresy, ve kterých se objevují klíčová slova, vztahující se k dané stránce. V implementovaném systému jsou tvořeny

na základě titulku stránky, před nímž je umístěn unikátní identifikátor k zabezpečení jedinečnosti vygenerované adresy.

Pro vyhledávače nemusí být jednoduché nalézt všechny existující stránky na serveru. Pro tento účel je vytvořena mapa webu, která mimo jiné poslouží návštěvníkům k rychlé orientaci. Mapa webu může být na základě požadovaného formátu jednak v podobě XHTML dokumentu, ale také v podobě protokolu *sitemap*[4]. Jelikož veškerý obsah redakčního systému je uložen v relační databázi, tyto mapy lze automaticky generovat bez nutnosti ruční údržby.

[4] *Sitemaps XML format*. [online] Dostupné z: http://www.sitemaps.org/protocol.php. Poslední aktualizace 27.2.2008

Závěr

Cílem této práce bylo navržení a realizace internetové prezentace se systémem pro správu obsahu podle požadavků společnosti GMV Martini CZ s.r.o. V teoretické části práce byla analyzována problematika tvorby internetových stránek a webových aplikací, na jejímž základě byly vybrány technologie, pomocí kterých je možné efektivně vytvářet CMS systémy internetových prezentací na profesionální úrovni.

Serverová část systému je postavena na MVC modelu a vhodně rozdělena do samostatných komponent, čímž je dosaženo vysoké modularity a pružnosti vůči případnému dalšímu rozšiřování. Administrační část systému je tvořena samostatnou aplikací, vyvinutou pomocí technologie Flex. Uživatelé mohou k této aplikaci přistupovat prostřednictvím webového prohlížeče s podporou zásuvného modulu Flash Player. Také mají možnost využít platformy Adobe AIR a tak k systému pro správu obsahu přistupovat prostřednictvím lokálně instalované klientské aplikace. Tento způsob obsluhy webové aplikace poskytuje správcům internetové prezentace vysoký komfort a podporu komplexních uživatelských rozhraní, na rozdíl od webového rozhraní.

Pro komunikaci mezi serverem a klientskou administrační aplikací je použito webových služeb, postavených na architektuře REST. Na rozdíl od podobně zaměřených technologií, jakými jsou systémy s podporou volání vzdálených procedur, není tento přístup zdaleka tak těžkopádný. Přístup ke vzdálenému systému je přímočarý a jeho rozhraní disponuje vysokou pružností vůči případným změnám. V neposlední řadě, možnosti těchto webových služeb nekončí pouze u použití klientské aplikace, ale mohou být v budoucnu také s výhodou využity pro případnou integraci s dalšími informačními systémy.

Pro naplnění opravdu kritického pohledu na navržený redakční systém je nutné podotknout, že tento přístup nemusí být vždy nejvhodnější. V absolutním měřítku takto navržený systém je jistě na vyšší úrovni, nežli systémy s holým webovým rozhraním. Na druhé straně tvorba webového rozhraní je jistě jednoduší. Odpadá tak nutnost tvorby komunikačních rozhraní mezi systémy. Větší komplikace s sebou ovšem přináší synchronizace stavu mezi klientskou aplikací a serverovým systémem, potažmo nutnost udržovat dvojí definice modelů.

Určité řešení bych v budoucnu viděl v implementaci mechanismů obsažených v knihovně ActiveResource na klientské straně systému. Tato knihovna je součástí frameworku Ruby on Rails a slouží k mapování REST zdrojů na objekty, obdobným způsobem jako v případě ORM knihovny ActiveRecord. Je tak zajištěno zautomatizované mapování REST entit na objekty v aplikaci. Pro podporu těchto mechanismů v klientské části postavené na technologii Flex, bude ovšem nutná implementace obdobné knihovny v programovacím jazyce ActionScript.

Seznam použitých zdrojů

[1] AHMED, Tariq, HIRSSCHI, Jon a ABID, Faisal. *Flex 3 in Action*. Greenwich (Connecticut) : Manning Publications, 2009. 576 s. ISBN 978-1-933988-74-0.

[2] ARMSTRONG, P. *Flexible Rails : Flex 3 on Rails 2*. Greenwich (Connecticut) : Manning Publications, 2008. 592 s. ISBN 1-933988-50-9

[3] CRANE, Dave, BIBEAULT, Bear a LOCKE, TOM. *Prototype and Scriptaculous in Action*. Greenwich (Connecticut) : Manning Publications, 2007. 542 s. ISBN 1-933988-03-7

[4] FLANAGAN, David. *JavaScript : Kompletní průvodce*. 2. aktualiz. vyd. Brno : Computer Press, 2002. 842 s. ISBN 80-7226-626-8.

[5] HERNANDEZ, Michael a VIESCAS, John. *Myslíme v jazyku SQL : tvorba dotazů*. 1. vyd. Praha : Grada Publishing, 2004. 380 s. ISBN 80-247-0899-X

[6] HILLYER, Mike. *Managing Hierarchical Data in MySQL*. [online] [cit. 2009-05-08] Dostupně z: http://dev.mysql.com/tech-resources/articles/hierarchical-data.html

[7] KOFLER, Michael. *Mistrovství v MySQL 5 : Kompletní průvodce webového vývojáře*. 1. vyd. Brno : Computer Press, 2007. 808 s. ISBN 978-80-251-1502-2.

[8] KUBÍČEK, Michal. *Velký průvodce SEO : Jak dosáhnout nejlepších pozic ve vyhledávačích*. Brno : Computer Press, 2008. 320 s. ISBN 978-80-251-2195-5

[9] LEDFORD, Jerri a TYLER, Mary. *Google Analytics 2.0.* Wiley Publishing, 2007. 339 s. ISBN 978-0-470-17501-9

[10] MARSHALL, Kevin, PYTEL, Chad a YUREK, Jon. *Pro Active Record : Databases with Ruby and Rails*. 1st edition. New York : Apress, 2007. 294 s. ISBN 978-1-59059-847-4.

[11] MUSCIANO, Chuck a KENNEDY, Bill. *HTML a XHTML : Kompletní průvodce*. 1. vyd. Brno : Computer Press, 2000. 639 s. ISBN 80-7226-407-9

[12] SCHMULLER, Joseph. *Myslíme v jazyku UML : knihovna programátora*. 1. vyd. Praha : Grada Publishing, 2001. 360 s. ISBN 80-247-0029-8.

[13] *Sitemaps XML format*. [online] Dostupné z: http://www.sitemaps.org/protocol.php. Poslední aktualizace 27.2.2008

[14] STANÍČEK, Petr. *CSS Kaskádové styly : Kompletní průvodce*. 2. vyd. Brno : Computer Press, 2003. 178 s. ISBN 80-7226-872-4.

[15] THOMAS, Dave, FOWLER, Chad a HUNT, Andy. *Programming Ruby : The Pragmatic Programmers' Guide*. 2nd edition. Raleigh (North Carolina) : Pragmatic Programmers, 2005. 833 s. ISBN 0-9745140-5-5

[16] THOMAS, Dave a HEINEMEIER HANSSON, David. *Agile web development with Rails*. 2nd edition. Raleigh (North Carolina) : Pragmatic Programmers, 2006. 719 s. ISBN 978-0-9776166-3-3.

[17] VACULNÝ, Jiří. *Agilní metodologie programování*. [online] [cit. 2009-05-08] Dostupné z: http://www.fi.muni.cz/usr/jkucera/pv109/2003/xvaculny.htm.

Seznam použitých zkratek

AJAX	Asynchronous JavaScript And XML
ASP.NET	Active Server Page .NET
CMS	Content Management System
CRUD	Create-Read-Update-Delete
CSS	Cascading Style Sheets
DBMS	DataBase Management System
DTD	Document Type Definition
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
MVC	Model-View-Controller
MXML	Magic eXtensible Markup Language
OOP	Object-Oriented Programming
ORM	Object-Relational Mapping
PHP	PHP: Hypertext Preprocesor
RDA	Rich Desktop Application
REST	Representational State Transfer
RIA	Rich Internet Application
SGML	Standard Generalized Markup Language
SDK	Standard Development Kit
SEO	Search Engine Optimization
SQL	Structured Query Language
UML	Unified Modeling Language
URL	Uniform Resource Locator
WWW	World Wide Web
WYSIWYG	What You See Is What You Get
XHTML	eXtensible HuperText Markup Language

XML	eXtensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

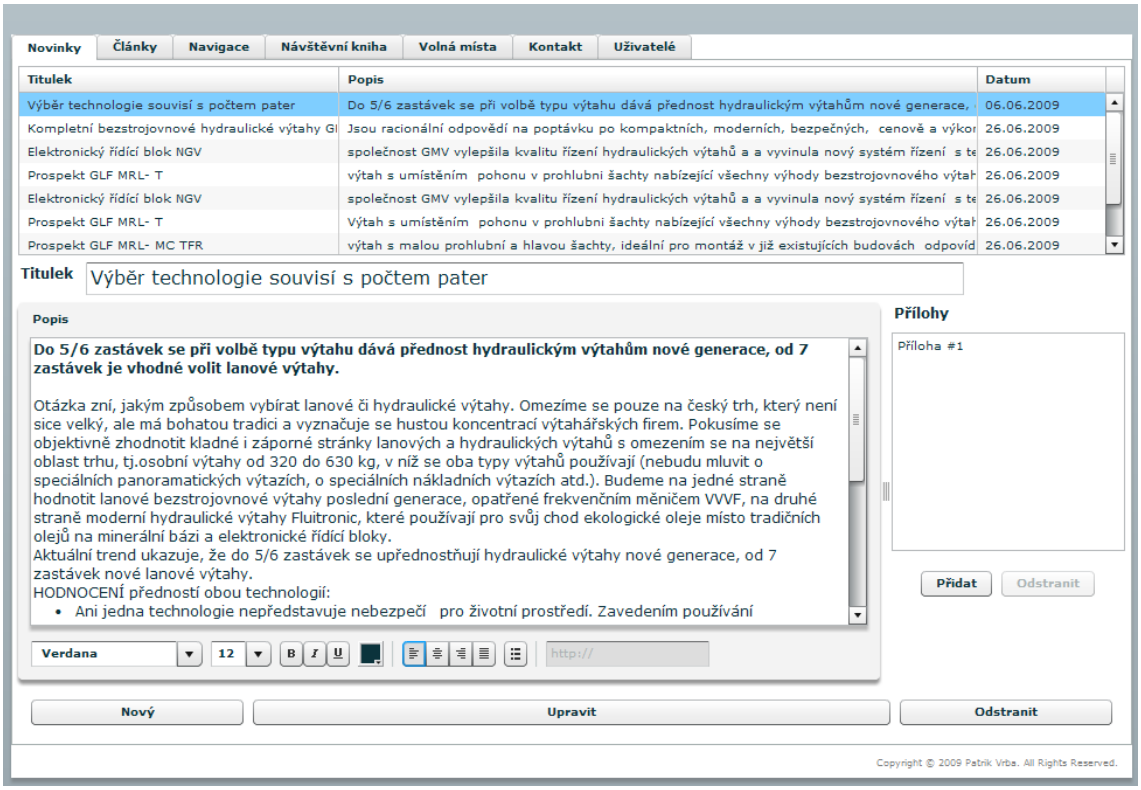
Seznam obrázků

Seznam tabulek

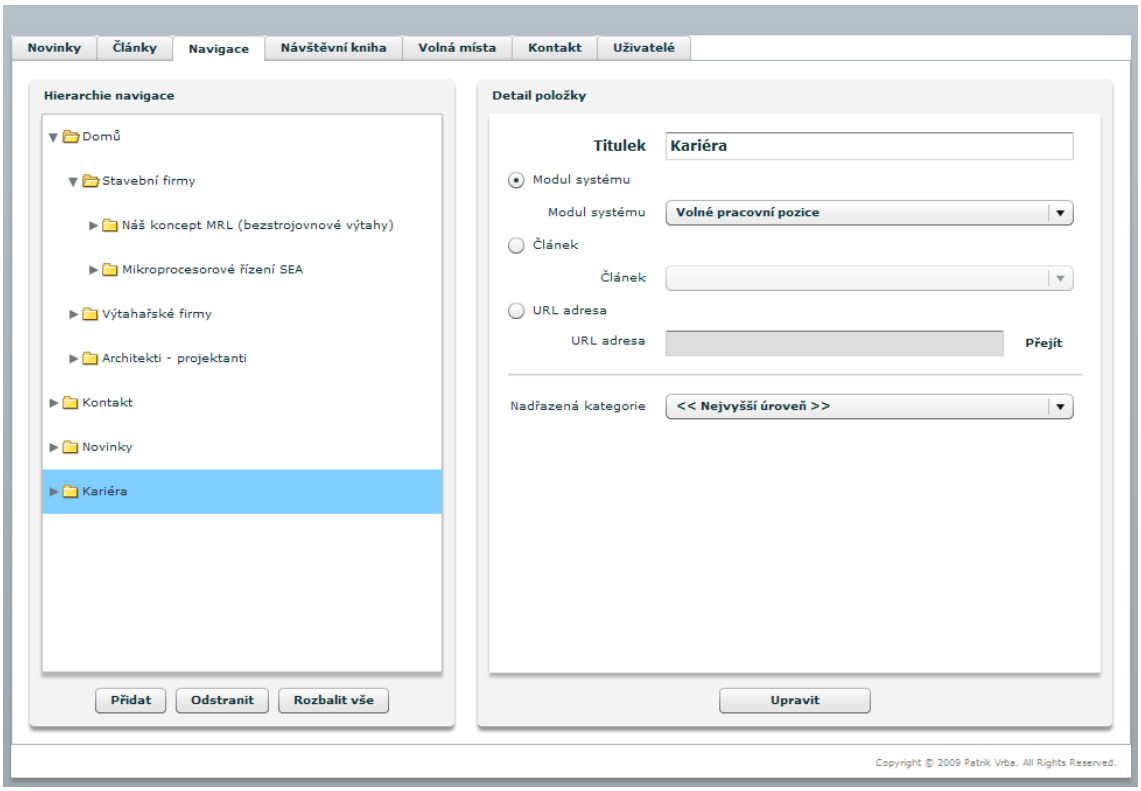
Seznam příloh

Přílohy

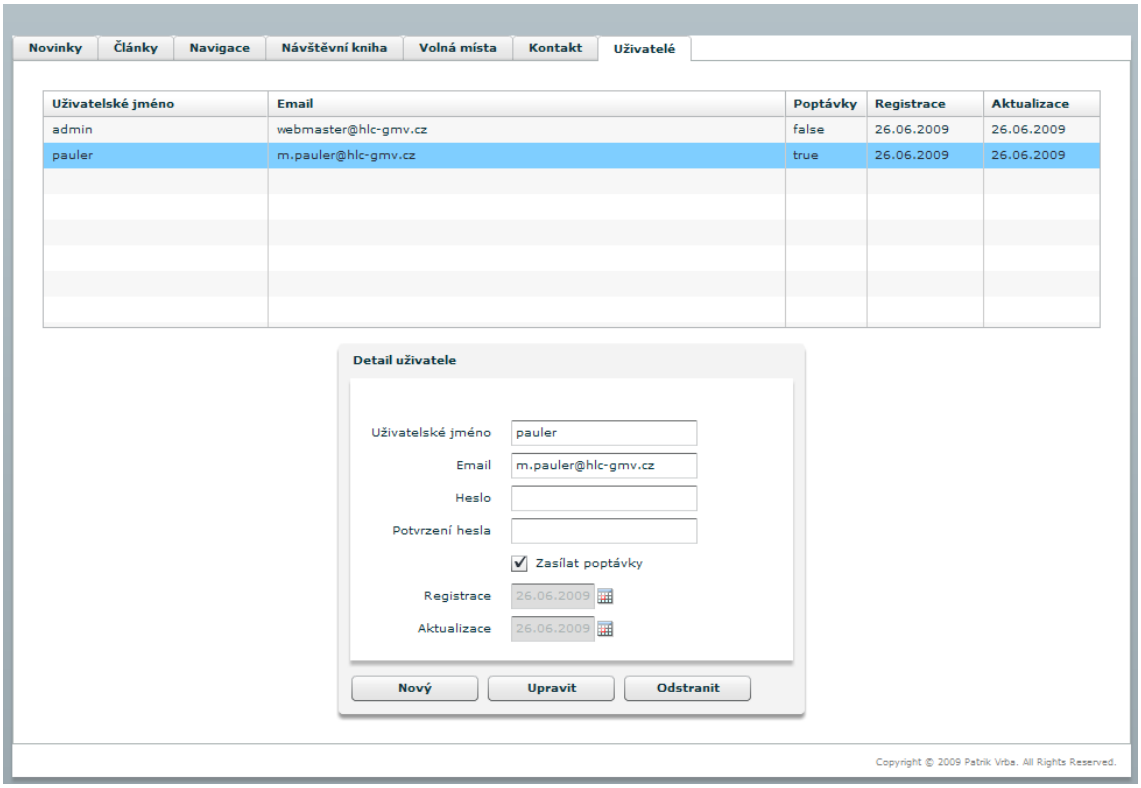
Příloha A - ukázka administračního rozhraní



Správa novinek



Správa navigační struktury webu



Správa uživatelských účtů

Příloha B – statický pohled na funkční model systému