VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

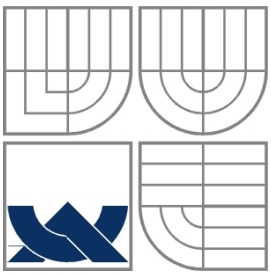
MODULÁRNÍ CMS PRO SPRÁVU WEBOVÉ PREZENTACE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

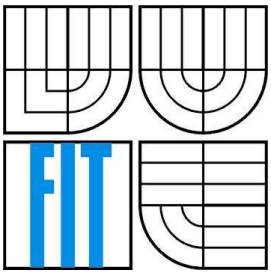
AUTOR PRÁCE
AUTHOR

Bc. DAVID DUŠEK

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODULÁRNÍ CMS PRO SPRÁVU WEBOVÉ PREZENTACE

MODULAR CMS SYSTEM FOR WEB MANAGEMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DAVID DUŠEK

VEDOUCÍ PRÁCE
SUPERVISOR

Mgr. ROMAN TRCHALÍK

BRNO 2012

Abstrakt

Náplní této práce je návrh a implementace systémů pro správu webové prezentace. Na začátku práce je definován pojem systému pro správu obsahu a jsou rozebrány stávající volně dostupné systémy tohoto typu. Následně jsou určeny požadavky na navrhovaný systém a to včetně aplikace, která umožní hromadnou centralizovanou správu verzí systému, provozovaného na různých doménách. V další části je vypracován návrh, podle kterého je systém naimplementován s použitím aplikačního rámce Nette a databázové vrstvy Doctrine.

Abstract

The content of this thesis is the design and implementation of system for managing website. At first we define term content management system and existing open-source systems of this type are discussed. Then we determinate the requirements for the designed system, including an application that enables centralized management of updates, operating on different domains. In the next part we design architecture of system and implement it with using Nette framework and database layer Doctrine.

Klíčová slova

CMS, systém pro správu obsahu, Nette, framework, návrhový vzor MVP, databázová vrstva Doctrine, ORM, hromadná aktualizace, koncept Hooků

Keywords

CMS, content management system, Nette, framework, design pattern MVP, database layer Doctrine, ORM, batch update, concept of Hooks

Citace

Dušek David, Modulární CMS pro správu webové prezentace, diplomová práce, Brno, FIT VUT v Brně, 2012

Modulární CMS pro správu webové prezentace

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Mgr. Romana Trchalíka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
David Dušek
27. července 2012

Poděkování

Děkuji svému vedoucímu panu Mgr. Romanu Trchalíkovi za odbornou pomoc při vypracování této diplomové práce.

Obsah

1 Úvod

Cílem této práce je návrh a implementace systému pro správu obsahu webové aplikace. V této úvodní kapitole si definujeme samotný pojem systému pro správu obsahu, zvláště pak systém pro správu webové prezentace. V druhé kapitole si rozebereme nejpoužívanější volně dostupné systémy určené pro správu webové prezentace za účelem získání přehledu o použité architektuře a technologiích. Následuje třetí kapitola, v které si definujeme funkční i nefunkční požadavky na vyvíjený systém, vycházející z předchozí části a praktických zkušeností, nabytých při práci s podobnými systémy. Na základě těchto požadavků provedeme ve čtvrté kapitole detailní návrh systému. V rámci této části si navrhneme architekturu systému pro správu obsahu, představíme mi použitý aplikační rámec Nette, databázovou vrstvu Doctrine a navrhneme fyzickou strukturu databáze. V následující kapitole se budeme zabývat implementací. Popíšeme si strukturu aplikace a s ní spojené změny v použitém aplikačním rámci, detailněji si popíšeme implementaci jádra systému i jednotlivých modulů. Navazující šestá kapitola je věnována prostředkům použitým pro testování a ladění ve fázi implementace. Poslední kapitola obsahuje náměty na další rozvoj systému a shrnuje dosažené výsledky.

Tato práce navazuje na semestrální projekt vypracovaný v rámci předmětu SEP. Z tohoto projektu jsou převzaty především části úvodu, popisu nejpoužívanějších systémů pro správu obsahu a analýzy požadavků.

1.1 Systém pro správu obsahu

CMS je zkratka z anglického *Content Management System*. Do češtiny je tento název překládán jako systém pro správu obsahu. Tato práce se zabývá návrhem systému pro správu obsahu webového, který bývá označovaného jako *Web-based* CMS. V češtině se pro tento druh CMS vžilo také označení redakční systém.

Obecně je CMS software, který poskytuje nástroje pro správu určitého druhu elektronického obsahu. Spravovaným obsahem mohou být textové dokumenty, obrázky, videa a další. Nad veškerým obsahem je většinou poskytována základní sada operací označovaná jako CRUD, z anglického *Create* (vytvoření/přidání), *Read* (čtení/zobrazení), *Update* (změna) a *Delete* (smazání) a případně další specifické funkce v závislosti na typu spravovaného obsahu.

Cílem systému pro správu obsahu je umožnit souběžnou práci se spravovaným obsahem více uživatelům najednou. CMS je vhodným prostředkem pro zefektivnění podnikové komunikace a automatizaci firemních procesů, protože uživatelům nabízí možnost přístupu k dokumentům a jejich editaci napříč podnikovou infrastrukturou. Dalším přínosem je pak možnost využití systému jako archivačního nástroje pro podnikové dokumenty.

Web-based CMS je systém pro správu webového obsahu. Většinou je implementován přímo jako webová aplikace a je provozován v prostředí webového prohlížeče. Hlavním jeho úkolem je správa obsahu veřejné části webu označované jako *front-end* prostřednictvím administrační části označované jako *back-end*, která je běžně přístupná pouze autorizovaným uživatelům. Na veřejnou část jsou kladeny nároky především s pohledu přístupnosti a použitelnosti webové prezentace a s tím spojenou SEO optimalizaci (*Search Engine Optimization* do češtiny překládané jako optimalizace pro vyhledávače). Funkčnost administrační části pak lze rozdělit do dvou částí. První část má za úkol správu obsahu webové prezentace, která zahrnuje správu stránek, článků, menu, multimediálních dat

a podobně. Druhá část se zabývá správou uživatelů včetně administrace řízení přístupů ke spravovaným zdrojům.

Web-based CMS jsou určeny především uživatelům, kteří si chtějí vytvořit vlastní webovou prezentaci, ale nemají patřičné znalosti v oblasti programování. Z toho plyne, že obsluha takového programu by měla být pokud možno co nejjednodušší a snadno pochopitelná bez hlubšího studia dokumentace nebo nutnosti absolvovat školení. Zároveň je nutné zohlednit uživatele, kteří CMS chtějí používat na profesionální úrovni a chtějí mít možnost, přizpůsobit jej svým požadavkům.

Cílem této práce je navrhnout *Web-based* CMS, které bude postaveno na modulární architektuře a díky tomu snadno rozšiřitelné, bude jednoduché na pochopení a ovládání i bez znalosti programování, nabídne podporu pro SEO optimalizace a umožní spravovat uživatele včetně přístupových práv. Navíc nabídne i podporu pro situace, kdy jeden subjekt spravuje více CMS na různých doménách a je potřeba efektivně šířit změny na všechny domény, které se mohou lišit (různé moduly, upravené části modulů a podobně).

2 Nejpoužívanější CMS

Tato kapitola shrnuje základní vlastnosti *Web-based* CMS. Pro tento účel byly vybrány tři nejpoužívanější *open-source* CMS [1] a to Wordpress, „Joomla!“ a Drupal. Postupně zde budou popsány z pohledu licenční politiky, systémových požadavků a architektury. V závěru kapitoly bude diskutována jejich podpora pro aktualizace a šíření změn.

2.1.1 Licence

Všechny tři CMS jsou vyvíjeny jako *open-source* projekty a jsou tedy dostupné zdarma. „Joomla!“ a Drupal jsou šířeny pod licencí GNU GPL (*General Public Licence*) a Wordpress pak pod licencí GPLv2. Přínos GPL licence z pohledu vývojáře je možnost prohlížení zdrojových souborů a v případě potřeby je i jejich úprava. Další výhodou je pak možnost bezplatného použití i pro komerční účely.

2.1.2 Systémové požadavky

Wordpress, Jommla!“ i Drupal jsou implementované s použitím skriptovacího programovacího jazyka PHP a jako databázový systém lze u všech použít MySQL. Pro svůj běh vyžadují aplikační server Apache nebo Microsoft IIS. Veškeré požadované technologie pro provoz diskutovaných CMS jsou shrnuta v následující tabulce (Tabulka 1).

	Wordpress 3.2	**Joomla! 1.7**	**Drupal 7**
PHP	PHP 5.2.4.	PHP 5.2.4.	PHP 5.2.5.
Databázový systém	MySQL 5	MySQL 5.0.4	MySQL 5.0.15 SQLite 3.3.7
Aplikační server	Apache Nginx	Apache 2.x Microsoft IIS 7	Apache 2.x Microsoft IIS
Další požadavky na moduly Apache	mod_rewrite	mod_mysql mod_xml mod_zlib	mod_rewrite

Tabulka 1: systémové požadavky [2], [3], [4]

2.1.3 Architektura

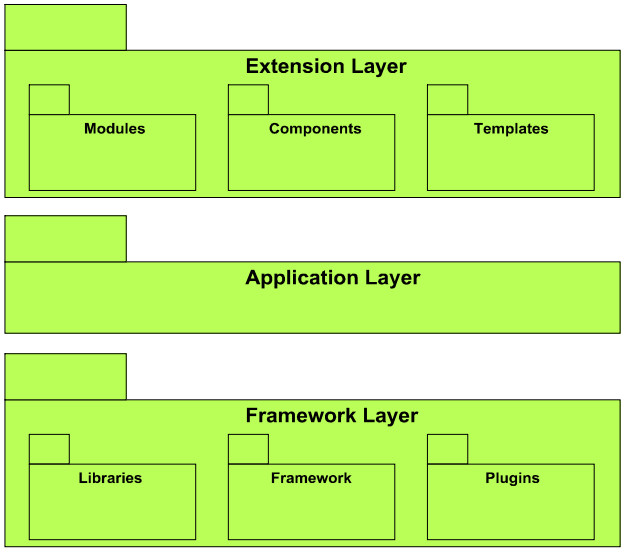
Nyní si podrobněji rozebereme architekturu vybraných systémů pro správu obsahu. Čerpáno bylo především z [5] a [6]

Wordpress

Wordpress byl původně navržen pro vytváření blogů a z toho částečně vychází i jeho architektura. Srdcem systému je jádro, které poskytuje *framework* pro rozšiřující *pluginy*. Jádro se vývojáři snaží optimalizovat, tak aby mělo pokud možno co nejmenší rozsah a vysokou rychlost. Wordpress používá pro rozšiřující moduly název *Plugin*. *Plugin* se k jádru připojuje pomocí funkcí nazvaných „*hooks*“ (háky). *Hooky* jsou funkce, které vyvolá jádro v určité fázi zpracování požadavku, při výskytu požadované události. Vyvolání *hooku* má za následek zavolání všech funkcí, které daný *hook* implementují. Wordpress lze také rozšířit pomocí vytvoření vlastního *Theme* (motivu vzhledu). Je to sada šablon a funkcí, která ovlivňuje výsledný vzhled vykreslené stránky.

Joomla!

„Joomla!“ je postavena na třívrstvé architektuře, která je zobrazená na následujícím obrázku (Obrázek 1). Nejnižší vrstvu tvoří vrstva *frameworku*, která je dále dělena na tři části a to vlastní *framework*, knihovny (*libreries*) a *pluginy*. Nad ní se nachází aplikační vrstva tvořená třídou dědící od třídy JApplication, která je součástí *frameworku*. V distribuci nalezneme tři samostatné aplikace a to aplikaci pro instalaci, pro správu administrační části a pro správu veřejné části webové prezentace. *Extension layer*, tedy vrstva rozšíření, je tvořena moduly (*Modules*), komponentami (*Components*) a šablonami (*Templates*). Komponenty jsou funkční celky, které mají jak administrační tak veřejnou část. Jako příklad komponenty může být komponenta pro články. Moduly by se daly chápat spíše jako bloky a slouží pro přidání funkčních částí do šablony stránky. Nakonec jsou tu šablony, které ovlivňují výsledný vzhled prezentace.



Obrázek 1: třívrstvá architektura CMS Joomla! [7]

Drupal

Drupal obsahuje, podobně jako Wordpress, jádro a je rozšiřitelný pomocí modulů. Moduly jsou s jádrem propojeny pomocí *hooků*. *Hook* má zde stejný význam jako u Wordpressu. Dále je Drupal možné rozšířit prostřednictvím *Theme* (motivu vzhledu), které ovlivňují vzhled veřejné části webu.

2.2 Aktualizace

Stávající systémy nabízí v tomto ohledu v podstatě dvě možné varianty aktualizace. První variantou je manuální nahrání nové verze přes FTP přímo na *hosting*. Před provedení této operace je doporučeno udělat zálohu starého systému včetně databáze a před nahráním nové verze smazat původní soubory.

Druhou variantou je automatická aktualizace, která se většinou provádí z administrační části CMS. I u automatizované aktualizace je u některých systémů třeba provést manuální zálohu databáze i samotného systému. Další nepříjemnou vlastností je, že aktualizace většinou týká pouze jádra systému a rozšiřující moduly musí být aktualizovány samostatně. Z důvodu oddělené aktualizace tedy vzniká riziko nekompatibility mezi jádrem a rozšiřujícími moduly.

Navrhované CMS by mělo řešit všechny popsané problémy spojené s aktualizací jádra i modulů systému. Navíc budeme požadovat, aby náš systém zvládal i hromadnou centralizovanou aktualizaci více domén naráz. Díky tomu bude pro správce snazší distribuovat novou funkcionalitu a opravy chyb na všechny svěřené domény.

3 Analýza požadavků

Tato kapitola se zabývá analýzou požadavků. V první části si stručně popíšeme běžné metody pro získávání požadavků. Další kapitola obsahuje výčet nefunkčních požadavků na systém. Následuje strukturovaný seznam funkčních požadavků rozdělený na požadavky kladené na jádro, administrační a veřejnou část CMS. V poslední části si definujeme požadavky na aktualizaci a udržovatelnost několika CMS spravovaných jedním subjektem.

3.1 Metody získávání požadavků

Získávání požadavků a jejich správné pochopení a interpretace představuje významný milník ve vývoji software produktu. Požadavky jsou získávány od zákazníka prostřednictvím zástupce zhotovitele, který nemusí mít znalost cílové technologie. Nebezpečí této fáze tkví v nedorozumění mezi zákazníkem a zástupcem zhotovitele z důvodů odlišného chápání pojmů jednotlivých odvětví. Z toho důvodu se často v této fázi sestavuje slovník pojmů. V následujících částech jsou stručně popsány klasické metody získávání požadavků. Tato kapitola čerpá z [8].

3.1.1 Rozhovor se zákazníkem

Rozhovor se zákazníkem si klade za cíl získat jak obecné informace o realizovaném software tak detailní specifikaci použití jednotlivých částí systému. Většinou je rozhovor složen ze strukturované a nestrukturované části. Nestrukturovaná část je spíše rozhovor o celkovém konceptu navrhovaného software, zatím co strukturovaná část se zaměřuje na detailní požadavky. Strukturovaná část může být vedena buď formou otázek, na které zákazník odpovídá bez navrhované odpovědi nebo si zákazník volí z nabízených možností. Snahou tazatele by mělo být hledat minimální možné řešením se zachováním všech stěžejních požadavků zákazníka. Rozhovor by však vždy měl být veden tak, aby zákazníkovi nebyly vnucovány myšlenky a nápady tazatele.

3.1.2 Dotazník

Dotazník je pasivní formou získávání požadavků, a proto nejsou jeho výsledky tolik relevantní. Většinou slouží jako doplněk k rozhovoru se zákazníkem. Výhodou získávání informací prostřednictvím dotazníků jsou menší vynaložené prostředky.

3.1.3 Studium stávajícího systému

Další metodou pro získávání požadavků je studium stávajícího systému. Tato metoda se může skládat vyzkoušení funkčnosti systému, prohlížení zdrojových souborů a studium dokumentace. Přínos této metody je především v získání doménových znalostí daného oboru.

Tato metoda byla použita i při stanovení požadavků pro navrhovaný systém. Trojice výše popsaných systémů pro správu obsahu je vyvíjena jako *open-source* a lze si je tedy stáhnout, vyzkoušet a prohlédnout si zdrojové soubory. Všechny projekty disponují i přehledně zpracovanou dokumentací.

Dalším zdrojem informací pro stanovení požadavků byla osobní zkušenost s prací ve firmě, zabývající se tvorbou a správou webových systémů. Právě tato praxe ukázala, jak je důležitá navrhovaná centralizovaná hromadná distribuce změn.

3.1.4 Pozorování prací u zákazníka

Poslední zde zmíněnou metodou sběru požadavků je pozorování prací u zákazníka. Tato metoda poskytuje především informaci o tom, jak zaměstnanci zákazníka se systémem pracují. Pozorování může být obecně vedeno dvěma způsoby:

- pasivně – pouze sledování zaměstnance zákazníka při práci
- aktivně – zástupce zhotovitele přímo se systémem pracuje

Nevýhodou této metody je, že pozorovaný zaměstnanec má snahu se chovat jinak, než kdyby pozorován nebyl.

3.2 Nefunkční požadavky

Systém bude provozován jako webová aplikace v prostředí webového prohlížeče. Administrační část by měla být optimalizována minimálně pro webové prohlížeče Firefox verze 3.5 a vyšší a Internet Explorer verze 7 a vyšší. Veřejná část by pak měla být optimalizována i pro prohlížeč Google Chrome verze 10 a vyšší. U administrační části se předpokládá korektní zobrazení při minimálním rozlišení 1280px x 1024px. Systém bude možné nainstalovat a provozovat na webovém serveru Apache 2. Jako databázový systém bude použito MySQL minimálně ve verzi 5. Jako implementační jazyk bude použito PHP 5.3 ve spojení s vhodně zvoleným frameworkem. Pro zefektivnění práce se systémem je doporučeno použití některého z JavaScriptových frameworků při vývoji uživatelského rozhraní.

Z bezpečnostních požadavků je vyžadován především přístup do administrační části systému pouze pro autorizované uživatele a možnost volit práva pro přístup k funkcím systému pro skupiny uživatelů. Dalším bezpečnostním požadavkem jsou zabezpečené formuláře jako ochrana před útoky typu *Cross-site Scripting* a podobnými.

3.3 Funkční požadavky

Součástí této části je definice funkčních požadavků na navrhované CMS. Požadavky jsou rozděleny do tří skupin a to na obecné požadavky na navrhovaný systém, požadavky na administrační část a požadavky na veřejnou část CMS. Požadavky vycházejí z praktické zkušenosti s některými volně dostupnými *Web-based* CMS a praxí v tomto oboru.

3.3.1 Obecné požadavky na systém

Řízení přístupu

Řízení přístupu bude řešeno na úrovni akcí prováděných v jednotlivých modulech. Uživatelé budou pro potřeby řízení přístupu zařazeni do uživatelských skupin. Systém by měl umožnit zakázat skupině také přístup k celému modulu. Pokud bude modul pro skupinu zakázaný, nebude zobrazen v administrační části webu. Pokud se uživatel pokusí přistoupit do části systému bez dostatečných oprávnění, bude přesměrován na formulář pro přihlášení a bude mu zobrazeno informační hlášení o nedostatečných právech pro přístup k požadované funkcionalitě.

Modularita

Systém bude možné rozšířit prostřednictvím modulů. Pro implementaci nového modulu by měla být dostupná šablona, nejlépe ve formě ukázkového modulu s minimální požadovanou funkcionalitou, potřebnou pro správné fungování modulu v rámci systému. Moduly na sobě budou nezávislé, respektive budou mít velice volnou vazbu, takže při odstranění libovolného modulu nedojde k chybě, která by způsobila nefunkčnost jiného modulu nebo dokonce celého systému.

3.3.2 Požadavky na administrační část

Multijazyčnost

Součástí každého modulu bude soubor/soubory, definovaného formátu, sloužících pro překlad modulu do požadovaného jazyka. Vždy bude přítomen alespoň jeden soubor s výchozím jazykem, kterým je čeština. Pokud nebude nalezen jazyk aktuálně používaný v rámci systému, použije se výchozí jazyková verze. Jazyk bude možné změnit v obecných nastaveních systému. Nastavení jazyka bude vázáno na přihlášeného uživatele, takže každý uživatel si bude moci zvolit vlastní jazyk.

Přihlášení do systému

Uživatelé se budou do systému přihlašovat pomocí dvojice uživatelské jméno a heslo. Uživatelské jméno bude unikátní v rámci systému a jeho unikátnost bude kontrolována při přidávání uživatele. Uživatelské jméno nebude možné změnit. Uživatelské jméno bude zadáváno jako samostatná položka a nebude tedy odvozeno z dalších požadovaných informací o uživateli jako například emailové adresy. Heslo může obsahovat velká a malá písmena bez diakritiky a číslice. Minimální délka hesla bude 6 znaků. Heslo nebude nikde uloženo v otevřené formě.

Struktura a navigace

Administrace bude rozdělena podle jednotlivých modulů, z nichž každý bude zastoupen položkou v menu. Aktivní modul bude mít submenu s akcemi, které modul nabízí. Menu bude generováno automaticky podle oprávnění uživatele k jednotlivým modulům a podle nastavení aktivity modulu.

Globální nastavení

Administrační menu bude obsahovat, kromě modulů, ještě speciální položku pro přechod do globálního nastavení sytému. V této části systému bude možné nastavit následující:

- nastavení výchozího jazyka
- výběr souboru s rozložením stránky pro administraci
- výběr souboru s rozložením stránky pro přihlašovací obrazovku
- výběr souboru s kaskádovým stylem pro určení vzhledu administrace

Nastavení výchozího jazyka se bude týkat uživatelů pouze v případě, že pro jimi zvolený jazyk nebude nalezen překlad požadovaného hesla nebo modul soubor s překladem pro zvolený jazyk nebude obsahovat vůbec.

Vzhled administrace bude určen kombinací zvoleného souboru s rozložením stránky a zvoleným kaskádovým stylem. Protože se obrazovka pro přihlášení může svým rozložením výrazně lišit od zbytku administrační části, bude moci uživatel zvolit pro tuto obrazovku jiný soubor s rozložením stránky. Pro určení vzhledu přihlašovací obrazovku bude použit stejný kaskádový styl jako pro zbytek administrace.

Modul pro správu uživatelů

Modul pro správu uživatelů bude neoddělitelnou částí systému. Modul bude složen z části pro správu uživatelů a uživatelských skupin a části pro správu přístupových práv.

Správa uživatelů

Uživatele bude moci přidat uživatel v roli administrátora nebo uživatel s oprávněním pro správu uživatelů. V systému bude po instalaci výchozí uživatel s uživatelským jménem `admin` a heslem `admin`. Pro přidání budou požadovány následující informace:

- jméno [text]
- příjmení [text]
- titul před jménem [text]
- titul za jménem [text]
- uživatelské jméno [text]
- heslo [text]
- uživatelská skupina [výběr, více možností]

Dále bude možné vypsat všechny uživatele. Seznam uživatelů bude možné filtrovat podle všech informací uvedených v předchozím výčtu s výjimkou hesla a titulů. Systém umožní přejít na detail uživatele a to jak z kompletního tak vyfiltrovaného seznamu. V detailu uživatele bude možné přímo editovat uložené informace z předchozího výčtu a provádět další akce, vztažené k uživateli, jako reset hesla, změna uživatelské skupiny a podobně. Při smazání uživatele je potřeba zachovat vazby na akce, které smazaný uživatel v systému vykonal.

Změna hesla

Změnu svého hesla může provést přihlášený uživatel v rámci administrační části systému. Resetovat heslo jiného uživatele, může pouze uživatel v roli administrátor nebo uživatel s dostatečným oprávněním pro správu uživatelů. Při resetu hesla dojde k vygenerování nového hesla, které se zobrazí uživateli, který tuto akci provedl a ten nové heslo sdělí majiteli daného účtu. Samotná operace předání nového hesla nebude součástí funkcionality tohoto systému.

Správa uživatelských skupin

Uživatele bude možné zařadit do skupiny sloužící pro řízení přístupu k jednotlivým částem systému. Uživatel může být zařazen vždy práv v jedné skupině. Pro vytváření nové skupiny bude požadován pouze její název. Uživatel by měl být po přidání nové skupiny automaticky přesměrován na stránku, kde bude možné přidat do vytvořené skupiny vybrané uživatele.

Editace skupiny umožní přidávat do skupiny uživatele a to i více najednou. Odebrání uživatele ze skupiny bude možné pouze tak, že bude přiřazen do jiné skupiny. Toto opatření zabrání situaci, kdy by uživatel nebyl zařazen do žádné uživatelské skupiny.

Smazání skupiny bude možné, pouze pokud nebude skupina obsahovat žádné uživatele.

Správa přístupových práv

Přístupová práva budou přidělována na úrovni uživatelských skupin. U každé skupiny bude volba pro editaci přístupových práv. Práva bude možné nastavovat na *povoleno* nebo *zakázáno* a to u každé akce v systému. Dále bude možné uživateli zakázat nebo povolit celý modul. Pokud nebude skupině povolena žádná akce s modulem, bude celý modul považovaný za zakázaný. Zakázaný modul nebude uživateli v administraci vůbec zobrazen.

Logování činnosti uživatele

Systém by měl nabídnout rozhraní, které umožní ukládat a následně zobrazovat aktivitu uživatele v systému. Rozhraní nabídne možnost přidání informace do logu s odkazem na spravovaný objekt, datum změny, identifikaci uživatele, který změnu provedl a případně poznámku s prováděnou změnou. Dále by mělo být možné získat log pro jednotlivé spravované objekty.

Modul pro správu modulů

Modul bude, stejně jako modul pro správu uživatelů, vždy v systému přítomen. Systém prostřednictvím modulu zobrazí seznam všech nalezených modulů. Ve výpisu bude zobrazen stav modulu (aktivní/neaktivní) a možnost stav modulu přepnout. Pokud bude modul neaktivní, nebude se zobrazovat v administračním menu a jeho funkce budou pro zbytek systému nedostupné.

Modul by měl nejlépe v rámci výpisu modulů nabídnout volbu pro změnu pořadí modulů, které by pak určovalo pořadí modulů v administrační části.

Modul pro správu menu

Tento modul bude sloužit pro správu menu pro veřejnou část webové prezentace. Každá položka menu bude obsahovat odkaz, přes který bude obsah zpřístupňovat a odkaz obsahu v rámci systému.. Modul bude zároveň implementovat funkce pro vygenerování mapy stránek a to ve formátu HTML pro zobrazení na webu a ve formátu xml se strukturou vhodnou pro vyhledávací roboty. Pro každé menu bude potřeba uchovávat následující položky:

- název [text]
- typ odkazovaného obsahu [výběr]
- odkaz na obsah [výběr]
- odkaz [text]
- pořadí [celé číslo]

Pokud bude menu aktivní, zobrazí se na stránce. Pořadí jednotlivých menu/položek bude určovat položka pořadí. Pokud nebude pořadí zadáno, bude menu seřazeno podle pořadí vložení do systému, kdy první bude menu/položka přidané jako první.

Modul stránky

Modul dovolí uživateli spravovat stránky zobrazované na veřejné části webu. Pro přidání nové stránky bude možné zadat základní informace o stránce:

- název [text]
- obsah [textové pole]
- titulek [text]
- popis [textové pole]
- klíčová slova [text]

Pro tvorbu obsahu stránky bude použit WYSIWYG editor. Ten by měl uživateli usnadnit práci ve smyslu tvorby odstavců, nadpisů a vkládání odkazů. Modul by měl nabízet rozhraní umožňující rozšiřovat funkcionalitu editoru z ostatních modulů. Například přidání volby vložit obrázek pokud bude aktivován modul pro správu obrázků.

Ve výpisu stránek bude možné nastavit jednu ze stránek jako domovskou a stránku publikovat. Systém by měl automaticky zajistit, že domovská stránka (*homepage*) bude v rámci CMS právě

jedna. Zároveň by systém měl zaručit, že stránka, která bude označena jako domovská, bude publikovaná.

Modul bude dále obsahovat samostatnou položku v submenu vedoucí na globální nastavení webové prezentace na veřejné části webu. Zde bude možné nastavit tyto položky:

- titulek [textové pole]
- popis [text]
- klíčová slova [text]
- soubor s layoutem [výběr]
- kaskádový styl [výběr]
- ikona [textové pole]

Položky titulek, popis a klíčová slova budou použita pro stránky, kde tyto údaje nebudou zadány. Toto opatření má význam především z pohledu optimalizace pro vyhledávače. Soubor s layoutem bude určovat rozložení prvků na stránce a systém automaticky vytvoří nabídku dostupných stylů podle dostupných souborů ve složce se styly. Kaskádový styl určuje výsledný vzhled stránky a bude podobně jako layout, vybrán z nabídky vytvořené systémem z dostupných souborů. Ikona bude zadána jako text, který bude reprezentovat relativní cestu k souboru s ikonou zobrazenou na záložce.

Modul obrázky

Obrázky bude možné sdružovat do galerií. Galerie bude moci obsahovat obrázky nebo další galerii/e. Uživatel bude mít možnost nahrát obrázky hromadně. Je požadována podpora minimálně formátů PNG, JPEG/JPG a GIF. Formát obrázku nebude potřeba před nahráváním specifikovat. O obrázku/galerii budou uloženy následující informace:

- název [text]
- popis [text]

Při hromadném nahrávání bude název obrázku automaticky nastaven na název galerie, do které bude obrázek nahráván nebo bude nastaven na slovo „Obrázek“, pokud bude vkládán do výchozí galerie. Při mazání galerie budou automaticky odstraněny i obrázky a galerie v ní obsažené.

Modul by měl nabídnout funkce, které umožní integrovat vkládání obrázků a galerií do obsahu stránky prostřednictvím WYSIWYG editoru. Dále by modul měl nabídnout vytvoření výpisu galerie jako samostatné stránky.

3.3.3 Požadavky na veřejnou část

Tvar adres

Adresy na veřejné části webu musí být optimalizované pro vyhledávače a měli by tedy být ve tvaru takzvaných *Cool URL*. *Cool URL* se nazývají adresy ve tvaru odrážejícím strukturu webu a čitelném pro běžného uživatele. Jako příklad si můžeme uvést adresu v klasickém tvaru www.fiktivnishop.cz?kategorie=ponozky&id=5 oproti www.fiktivnishop.cz/ponozky/5 [9].

Validnost generovaných stránek

Stránky generované systémem by měli být validní. Validita kódu bude ověřována pomocí Markup Validation Service organizace W3C [10]. Kód stránek bude možné označit jako validní pokud validátor nenalezne žádnou chybu v kódu stránky. Varování jsou akceptovatelná. Tento požadavek se týká pouze stránek generovaných přímo systémem a ne obsahu generovaného knihovnami třetích stran.

Šablony vzhledu

Výsledný vzhled prezentace bude záležet na nastavení layoutu dané stránky a zvoleným souborem s kaskádovým stylem. Toto nastavení by mělo být dostupné jak pro celou webovou aplikaci, tak pro každou stránku samostatně.

3.4 Požadavky na aktualizaci CMS

CMS bude možné aktualizovat dvěma způsoby. První způsob bude určen pro případ, kdy je prostřednictvím CMS spravována pouze jedna doména. Pro tento případ bude možné aktualizovat systém přímo z administračního rozhraní. Aktualizace proběhne automaticky včetně vytvoření zálohy stávajícího CMS a databáze. Při aktualizaci bude možné zvolit, které moduly se mají aktualizovat. Soubory pro aktualizaci se budou stahovat z vývojářského SVN. Adresu SVN bude možné změnit před každou aktualizací.

Druhý způsob bude určen pro situaci, kdy subjekt spravuje několik domén, které mohou být rozmístěny na různých serverech. Aplikace by měla nabídnout možnost nastavení přístupových informací pro jednotlivé domény. U každé domény bude možné zvolit šablonu pro aktualizaci, která bude určovat, moduly se na dané doméně budou aktualizovat. Jedna šablona bude moct být sdílena mezi více doménami. Soubory pro aktualizace budou moct být opět získávány z vývojářského SVN. Adresu SVN bude možné nastavit pro každou šablonu samostatně. Při aktualizaci se opět na každé doméně automaticky vytvoří záloha stávajícího CMS a databáze. Aplikace umožní zrušit poslední aktualizaci a vrátit se tak na předchozí verzi systému.

Další variantou pro hromadnou aktualizaci by mohl být systém, který by umožňoval kontrolu aktualizace na úrovni zdrojových souborů. Systém by pracoval na podobném principu jako SVN. Při požadavku na aktualizaci by se proti sobě porovnaly předchozí verze systému s aktuální verzí a podle rozdílů by se určili odpovídající řádky v jednotlivých souborech na dané doméně, které by měli být aktualizovány. Vzniklé konflikty by pak uživatel řešil v rámci aplikace manuálně.

4 Návrh

Cílem této kapitoly je navrhnout systém podle požadavků specifikovaných v předchozí kapitole. V první části se zaměříme na popis architektury navrhované aplikace. Postupně se seznámíme se zvoleným aplikačním rámcem Nette a databázovou vrstvou Doctrine 2. Na závěr této části se seznámíme s konceptem *hooků* a navrhneme strukturu modulu. V další části prodiskutujeme návrh systému pro hromadnou aktualizaci a kapitolu zakončíme návrhem databáze.

4.1 Návrh architektury

Kvalitní návrh architektury je pro vývoj softwarového projektu stěžejní. Cílem navrhované architektury by měla být možnost dekompozice systému na menší nezávislé celky, která přináší usnadnění práce v navazujících fázích vývoje tedy v implementaci, testování a údržbě.

Protože námi navrhovaná aplikace má být modulární, zvolíme architekturu s jádrem a moduly. Jádro systému bude zajišťovat integraci modulů a bude sloužit jako prostředník pro styk těchto modulů. Jádro bude nabízet množinu standardních funkcí, které budou tvořit rozhraní pro integraci s moduly. Díky tomu budou moduly na sobě vzájemně nezávislé, snadno testovatelné a udržovatelné. Zároveň se zaručí snadná rozšiřitelnost aplikace pomocí přidání nových modulů.

Pro tento projekt je požadováno použití vhodného aplikačního rámce neboli *frameworku*. S přihlédnutím k požadavkům na běh aplikace na webovém serveru s podporou PHP 5.3, zvolíme *framework* Nette. Tento aplikační rámec je postavený na třívrstvém architektonickém vzoru MVP, který bude popsán v následující sekci věnované Nette. Protože samotný *framework* nenabízí databázovou vrstvu, která by podporovala ORM (*Object-relational Mapping*), zvolíme si pro tento účel projekt Doctrine.

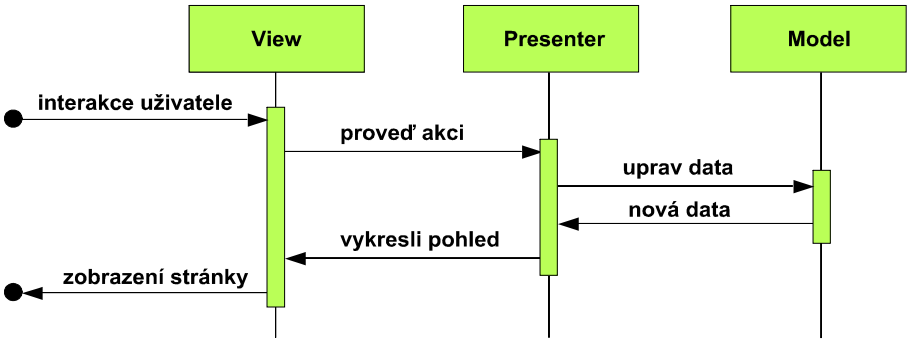
4.1.1 Nette framework

Nette [11] je aplikační rámec pro vývoj aplikací ve skriptovacím jazyce PHP. Vydávány jsou tři distribuce a to pro starší specifikaci PHP 5.2 a novější PHP 5.3. Pro PHP 5.2 jsou vydávány dvě odlišné distribuce a to varianta s použitím prefixů a varianta bez prefixů.

Pro náš projekt použijeme distribuci určenou pro PHP 5.3. Prvním důvodem k tomuto rozhodnutí je předpoklad, že právě tato distribuce jakožto nejnovější, bude dále vyvíjena, zatímco od vývoje distribucí pro verze PHP 5.2 se bude postupně upouštět. Dalším důvodem je pak podpora jmenných prostorů v PHP 5.3. Protože vyvíjíme poměrně rozsáhlý webový informační systém, počítá se s integrací knihoven a projektů třetích stran. Při takové integraci hrozí riziko kolize jmen identifikátorů a tříd. Deklarací jmenných prostorů lze dosáhnout oddělení jednotlivých částí aplikace a vyhnout se tak těmto problémům.

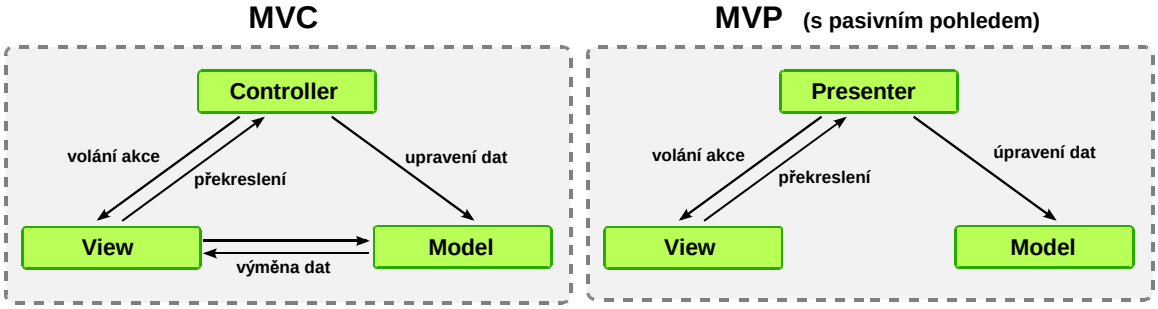
Aplikace vytvářené prostřednictvím Nette jsou postaveny na třívrstvém architektonickém vzoru MVP (*Model-View-Presenter*). Vrstva modelu zastřešuje práci s daty a potažmo s databázovým systémem. Pohled (*View*) je vrstva sloužící pro prezentaci dat uživateli. Protože navrhovaný systém bude pro prezentaci dat používat webový prohlížeč, je výsledkem této vrstvy webová stránka ve značkovacím jazyce HTML případně XHTML. Poslední vrstvou je Presenter. Tato vrstva obsahuje veškerou business logiku aplikace a stará se o zpracování požadavku a odeslaní odpovědi.

Zpracování požadavku pak probíhá tak jak je ukázáno na následujícím obrázku (Obrázek 2). Uživatel provede interakci s pohledem, například stisknutí tlačítka pro odeslání formuláře. Pohled v reakci na událost zavolá příslušnou akci presentru. Presenter provede kód akce a v případě potřeby interaguje s modelem za účelem změny dat. Presenter vytvoří odpověď na základě provedené akce a změněných dat a předá jí pohledu. Pohled se překreslí a zobrazí výsledek provedené akce uživateli.



Obrázek 2: zpracování požadavku v MVP

V případě Nette se jedná o variantu návrhového vzoru MVP s pasivním pohledem [12]. Hlavním rozdílem mezi tímto návrhovým vzorem a více známým MVC (Model-View-Controller) je v existenci vazby mezi pohledem a modelem. Zatímco u MVC tato vazba existuje a pohled může získávat data přímo z modelu, u diskutované varianty MVP tato vazba vůbec neexistuje a pohled získává data pouze prostřednictvím presentru. Výhodou tohoto přístupu je snazší testování aplikace, protože veškerá logika aplikace je součástí presentru. Vazby mezi vrstvami v obou návrhových vzorech ukazuje následující obrázek (Obrázek 3).



Obrázek 3: rozdíl mezi návrhovým vzorem MVC a MVP [13]

Vrstva pohledu v Nette je zastoupena šablonovacím systémem Latte. Tento šablonovací systém poskytuje prostředky pro definici bloků, dědičnost šablon a základní makra pro výpis proměnných, větvení, iteraci, vložení bloku kódu a podobně.

Dřívější distribuce *frameworku* neobsahovala podporu pro vrstvu modelu a bylo nutné použít externí knihovnu. K tomuto se nejčastěji používala databázová vrstva Dibi, pro jejíž integraci byl *framework* připraven. Nyní je součástí distribuce i databázová vrstva s názvem Database. Ta je však implementací pouze vrstvy DBAL (*Database Abstraction Layer*), která poskytuje odstínění od konkrétního typu použitého databázového systému, ale již neobsahuje funkčnost vrstvy ORM (*Object-relational Mapping*). Proto pro tento projekt použijeme databázovou vrstvu Doctrine.

Další důležitou novinkou použité verze Nette je použití *Dependency Injection* a systémového kontejneru. Kontejner slouží pro vytváření a následné zpřístupnění služeb. Systémový kontejner je vytvořen při startu aplikace na základě konfiguračního souboru. Jako službu si můžeme představit například připojení k databázi. Systémový kontejner je dostupný v celé aplikace a nahrazuje tak

potřebu globálních proměnných. *Dependency Injection* by se dalo přeložit jako vkládání nebo předávání závislostí. Jedná se o techniku pro sestavování složitých objektů a jejich závislostí. Pokud třída potřebuje ke své činnosti nějaké další třídy/služby, jsou jí obstarány objektem, který vytváří její instanci. Samotná třída se tedy o „shánění“ závislostí nemusí starat. Existují tři různé způsoby vložení závislostí. Prvním z nich je vložení závislostí v konstruktoru třídy. Právě tento způsob využívá Nette. Zbývající dva jsou pak pomocí metod *set* dané třídy nebo prostřednictvím rozhraní. [14]

4.1.2 Databázová vrstva Doctrine

Použitý databázový systém MySQL je relační a použitý programovací jazyk PHP je objektový. Narážíme tedy na problém mapování dat mezi těmito dvěma schématy. Problém řeší vrstva databázového modelu ORM. Protože použitý aplikační rámec neobsahuje implementaci této vrstvy, použijeme projekt Doctrine.

Doctrine je určena pro PHP verze 5.3 a vyšší. Jedním z hlavních důvodů proč nelze použít starší verze PHP je použití jmenných prostorů. Pro mapování mezi objektovým a relačním schématem je použita implementace návrhového vzoru *Data Mapper*. Dřívější implementace Doctrine používala návrhový vzor *Active Record*. Všechny dotazy a práce s entitami jsou centralizované do jednoho místa a tím je třída `EntityManager`. Tato třída vnitřně používá implementaci návrhového vzoru *Unit of Work*. Jeho cílem je snížení počtu dotazů na databázi, čehož dosahuje ukládáním stavu entit a odesíláním změn do databáze až v okamžiku dokončení celé operace. Entity se definují pomocí komentářových anotací a díky čemuž může být jako entita použit objekt libovolné třídy. Doctrine se skládá ze tří samostatných vrstev a to DBAL, *Common* a ORM.

Nejnižší vrstva *Common* obsahuje nástroje pro práci s kolekcemi, anotacemi a cachováním, třídu pro autoloading a další nástroje, které jsou využívány vyššími vrstvami. Najdeme zde také abstraktní třídy, které jsou ve vyšších vrstvách použity pro odvozování tříd nových.

DBAL (*Database Abstraction Layer*) zajišťuje abstrakci od konkrétního typu použitého databázového systému. Vrstva rozšiřuje standardní PHP databázové *drivery* implementující rozhraní PDO (*PHP Data Objects*). Dále tato vrstva definuje a implementuje operace pro speciální dotazovací jazyk DQL (*Doctrine Query Language*).

Nejvyšší vrstva je pak ORM (*Object-Relational Mapping*). Tato vrstva má na starost mapování mezi relační databází a objektově orientovaným jazykem. Dále poskytuje metody pro práci s entitami a umožňuje dotazování pomocí dotazovacího jazyka DQL. [15]

4.1.3 Vazby modulů

Jedním z požadavků na architekturu systému je modularita. Nové moduly by měli jít do systému snadno přidávat a případná absence modulu by neměla způsobit chyby v jiných modulech nebo dokonce selhání celého CMS. Hlavní důraz je tedy potřeba klást na nízkou vazbu mezi moduly. Zároveň je žádoucí komunikace mezi jádrem a moduly a moduly navzájem.

Pro řešení tohoto požadavku použijeme koncept *hooků*. Koncept je založen na jádře, které nabízí rozhraní pro registraci a spouštění *hook* metod. Jakékoli volání metody je pak realizováno právě přes toto jádro, které v případě nalezení požadované metody ve svém registru, vyvolá zpracování této metody na všech instancích tříd, které metodu registrovaly. Pokud není požadovaná metoda nalezena, je vrácena předdefinovaná hodnota. Tento koncept tedy zaručí, že nebude volána neexistující metoda nebo metoda nad instancí třídy, která v systému neexistuje. *Hooky* používá například i již zmiňované *open-source* CMS Drupal. Implementační detaily tohoto konceptu budou dále popsány v kapitole věnované implementaci.

4.1.4 Struktura modulu

Za modul bude považována složka, která bude pojmenována podle názvu modulu a to s prvním malým písmenem. Obsah složky pak bude obsahovat předepsanou hierarchii složek a souborů. Pro vytvoření nového modulu, bude v systému vzorový modul (adresář), který bude obsahovat minimální potřebné soubory pro bezproblémovou integraci modulu do systému. Nyní se podívejme na příklad modulu s názvem user:

`/user`

`/languages`	Soubory s překlady. Každý jazyk bude uložen v samostatném souboru. Jméno souboru bude tvořeno dvouznakovou zkratkou jazyka podle ISO 639-1 a bude mít příponu `.lang`. Obsah souboru pak bude tvořit jednorozměrné pole v JSON notaci ve tvaru klíč hesla a překlad hesla.
`/templates`	*Template* soubory pro jednotlivé *presentery*.
`/models`	Třídy spojené s databázovou vrstvou. Zde budou nejčastěji třídy entit pro Dotrine.
`/presenters`	*Presentry* použité pro administrační část daného modulu.
`/templates`	*Template* soubory šablonovacího systému Latte pro akce specifikované v rámci *presenterů*.
`/UserModule.php`	Třída modulu, kde název je složen ze jména modulu s prvním velkým písmenem a slova `Module`. V tomto souboru budou specifikovány *hook* metody, na které je tento modul schopen reagovat a případně další pomocné funkce například pro *presentry*.
`/config.ini`	Konfigurační soubor se základními informacemi o modulu jako verze nebo autor.
`/create.sql`	Skript SQL, který se provede při prvním zavedení modulu. Tento soubor není povinný.

4.2 Architektura aplikace pro hromadnou aktualizaci

V této části využijeme vhodně zvolenou architekturu systému a aplikaci pro hromadnou aktualizaci budeme realizovat jako rozšiřující modul do realizovaného CMS. Z požadavků plyne potřeba správy uživatelů a jejich přihlášení do systému. Tuto funkcionalitu již máme implementovanou jako součást modulu pro správu uživatelů. Dále máme k dispozici modul pro lokální aktualizaci, který nabízí požadovanou funkcionalitu pro aktualizaci systému na dané doméně. Pokud tedy tento modul rozšíříme o možnost vzdálené konfigurace a spuštění procesu aktualizace, zjednoduší se nám značně požadavky na modul pro hromadnou aktualizaci. Ten bude implementovat pouze vytváření a správu aktualizačních šablon, domén a vzdálené volání modulů pro lokální aktualizaci. Díky tomu dosáhneme distribuce zátěže linky i vlastního serveru, protože celý proces aktualizace si bude každá doména řídit samostatně.

4.3 Návrh schématu databáze

V této části se zaměříme na návrh fyzické struktury databáze navrhovaného systému. Při návrhu musíme mít na paměti požadavek modulárnosti systému. Z toho plyne nutnost oddělit i tabulky jednotlivých modulů, tak aby při odstranění některého z nich, nedošlo k porušení referenční integrity databáze. Výsledný návrh fyzického schématu databáze je v příloze B. Tabulky jsou seskupeny podle modulů, kterým náleží.

V modulu pro správu uživatelů potřebujeme uchovávat informace o uživatelích, přístupových právech, uživatelských skupinách a záznam aktivity uživatelů. Důležitý fakt při návrhu tabulky pro uživatele hraje vazba na tabulku `user_log`. Ta bude sloužit k uchovávání změn spravovaného obsahu. Každá změna obsahu se přitom váže na konkrétního uživatele a měla by bát dostupná i v případě odstranění uživatele z databáze. Proto zavedeme atribut `deleted`, který bude sloužit k logickému odstranění uživatele. Takový uživatel nebude mít k systému přístup, ale informace o něm budou dále dostupné. Stejný přístup je pak možné použít i ostatních entit, které nelze z databáze fyzicky odstranit, ale potřebuje přístup k jejich datům. V tabulce `user_log` si všimněme atributů `entity` a `entity_id`, které slouží právě pro odstranění vazby mezi tabulkami různých modulů. Tento koncept bude detailněji popsán dále v kapitole zabývající se implementací modulu pro správu uživatelů. Tabulka `user_group` je určena pro ukládání informací o uživatelských skupinách a spolu s tabulkami `user_group_operation`, `user_operation` a `user_resource` bude použita pro řízení přístupu.

Modul pro správu stránek, bude ke své činnosti potřebovat ukládat data o nastavení webové prezentace a obsahu jednotlivých stránek. Stránkou však nemusí být pouze webová stránka, ale může to být například i galerie obrázků. Pro takové případy si tedy vyčleníme obsah webové stránky do zvláštní tabulky `page` a obálky se základními informacemi o stránce a odkaz na příslušný obsah budeme ukládat do tabulky `page_content`.

Tabulka `menu` bude sloužit pro potřeby ukládání informací v modulu pro správu menu. Aby bylo možné vytvářet i hierarchická menu, obsahuje tabulka cizí klíč `parent`, který je odkazem do stejné tabulky.

Modulu pro správu modulů postačí pro jeho činnost také jediná tabulka, v které budou u každého modulu uloženy informace získané z příslušného konfiguračního souboru. Pro potřeby řazení modulů v menu administrační části webu bude použit atribut `order`. Atribut `active` bude použit pro aktivaci a deaktivaci modulu a atribut systém pro odlišení systémových a uživatelských modulů.

V modulu pro správu obrázků potřebujeme ukládat informace o obrázcích a galeriích. Aby bylo možné zařazovat obrázky do galerií, přidáme k tabulkám `image`, `image_gallery` ještě třetí tabulku `image_gallery_image`. Díky posledně jmenované vazební tabulce, bude možné jeden obrázek zařadit i do více galerií současně.

Modul pro lokální aktualizace potřebuje ukládat informace o historii provedených aktualizací. K tomuto účelu slouží tabulka `update`. Nejdůležitějším atributem je atribut `bckup_dir`, který nese informaci o složce v které je uložena záloha systému před provedením dané aktualizace. Tento atribut bude důležitý především pro obnovení systému na některou z předchozích verzí.

Poslední skupinou jsou tabulky určené pro modul hromadné aktualizace. Z požadavků plyne potřeba zařazování domén do aktualizačních šablon. Tabulka `updater_template` tedy ponese informace o jednotlivých šablonách a zařazení domény do šablony pak bude zajišťovat atribut `template_id` v tabulce `updater_domain`. Zbývající dvě tabulky `updater_update` a `updater_update_domain`, budou sloužit pro uložení záznamu o provedení aktualizace, podobě jako tomu bylo u modulu pro lokální aktualizaci.

5 Implementace

Cílem této kapitoly je důkladně popsat implementaci systému podle návrhu z předchozí kapitoly. V první části si vyjmenujeme a stručně popíšeme nástroje použité pro implementaci a testování systému. Následně se zaměříme na strukturu aplikace, kde si ukážeme rozdíly mezi strukturou požadovanou v rámci Nette *frameworku* a výslednou strukturou pro vyvíjenou aplikaci. Dále se již budeme zabývat vlastní implementací. Jako první se zaměříme na změny v souborech *frameworku*, vyžadované právě odlišnou strukturou aplikace. Následovat bude rozsáhlejší kapitola o jádru systému s podrobným popisem implementace *hooků*. V rámci této kapitoly si také představíme návrhový vzor *singleton*. Druhou rozsáhlejší částí bude popis implementace a funkčnosti dílčích modulů systému, kde se zaměříme především na modul pro aktualizaci systému.

5.1 Použité softwarové prostředky

Jako vývojový jazyk jsme si zvolili PHP. Pro vývoj na lokální stanici potřebujeme nainstalovat PHP, databázový systém MySQL a aplikační server Apache. Aby nebylo potřeba součinnost těchto tří komponent manuálně konfigurovat, použijeme předpřipravený balík s názvem WAMP. Pro snazší odstraňování chyb ve fázi implementace, si do PHP přidáme rozšíření Xdebug. Více o tomto rozšíření v kapitole 6.3. Dále nainstalujeme *framework* pro jednotkové testování PHPUnit, který je diskutovaný v kapitole 6.2. Součástí balíku WAMP je i nástroj pro správu MySQL databáze phpMyAdmin. Jedná se o program napsaný v PHP a provozovaný v prostředí webového prohlížeče. Pro naše potřeby však zvolíme desktopovou aplikaci Toad for MySQL. Aplikace nabízí širší funkcionalitu a pohodlnější práci s databází než je tomu u aplikace webové. Protože bude vyvíjený systém využívat aktualizaci z SVN serveru, musíme zajistit přenos zdrojových souborů prostřednictvím SVN protokolu na tento server. Pro tyto účely použijeme program TortoiseSVN. Nyní zbývá už jen volba vývojového prostředí. V tomto případě jsme se rozhodli pro vývojové prostředí NetBeans. K volbě právě tohoto vývojového prostředí nás vedly následující faktory: podpora pro vývoj v jazyce PHP, plugin pro tvorbu aplikací založených na frameworku Nette, a plná integraci s použitým ladícím rozšířením Xdebug a testovacím *frameworkem* PHPUnit.

5.2 Zvolená struktura aplikace

Na začátek si ukažme jak, vypadá předepsaná struktura Nette aplikace a stručně si popíšeme nejdůležitější části. Následující výčet není kompletní, ale postihuje nejdůležitější soubory a adresáře použitého *frameworku*.

`app`	
`/presenters`	*presentery* aplikace
`/templates`	šablony pro jednotlivé akce *presenterů*
`/bootsrap.php`	spuštění Nette aplikaci (vytvoření systémového DI kontejneru)
`/config.neon`	konfigurace aplikace
`doc`	adresář určený pro uživatelskou dokumentaci
`libs`	v tomto adresáři jsou vlastní soubory Nette *frameworku* a případně dalších použitých knihoven
`log`	záznamy o chybách vzniklých při běhu aplikace
`temp`	složka pro *cachování*
`www`	kořenová, veřejně dostupná složka, kde jsou uloženy *javascriptové* soubory, obrázky atd.
`/index.php`	všechny požadavky jsou pomocí souboru `.htaccess` směrovány na tento skript

Důvodem proč musíme pro naši aplikaci zvolit jinou strukturu, je především umístění *presenterů* a k nim příslušejících šablon (*templatů*). Protože chceme dosáhnout stavu, kdy bude modul reprezentován jediným adresářem, je nutné *presentery* a šablony z předepsaných adresářů vyčlenit. V předepsaném adresáři pro *presentery*, zanecháme pouze třídy sloužící k odvozování *presentrů* a v adresáři se šablonami pouze soubory rozložení (*layouty*) administrační části CMS. V adresáři `app` pak vytvoříme nový adresář `modules`, v kterém budou umístěny jednotlivé moduly.

Jádro systému umístíme do adresáře `libs` určeného pro knihovny. Zde vytvoříme adresář `CMS`, který bude obsahovat třídy jádra systému.

Poslední úpravou pak bude umístění veřejně dostupných souborů CMS do speciálního adresáře `www/CMS`. Tento adresář bude obsahovat kaskádové styly, obrázky, javascript soubory a javascriptové knihovny určené pro potřeby administrační části systému.

5.3 Potřebné úpravy Nette frameworku

V našem projektu, chceme používat upravenou adresářovou strukturu popsanou v předchozí části, která se liší od adresářové struktury podporované *frameworkem*. Z tohoto důvodu by *framework* nepracoval správně, a proto je potřeba provést některé změny přímo ve zdrojových souborech Nette.

První problém který je potřeba řešit, nastane při načítání *presenterů*. *Presentery* jsou automaticky hledány v adresáři `app/presenters` z kterého jsme je přesunuli. Musíme tedy změnit třídu `PresenterFactory`, která je součástí Nette *application* a má na starost právě vytváření instancí *presenterů*. Úprava spočívá ve vytvoření nové třídy `PresenterFactory` v adresáři `libs/CMS` a jmenném prostoru naší aplikace. Tato nová třída zdědí původní třídu a bude obsahovat patřičně přepsané metody `getPresenterClass`, `formatModulPresenterFile`, `formatPresenterClass`, a `unformatPresenterClass`, tak aby vyhledávali *presentery*

v nové struktuře. Nakonec ještě musíme třídu zaregistrovat v konfiguračním souboru `config.neon`, aby jí *framework* použil namísto původní.

Další nutnou úpravou, plynoucí ze zvolené adresářové struktury, je přepsání metody `formatLayoutTemplateFiles` ve třídě `BasePresenter`, která bude použita jako předek všech ostatních *presenterů*. Standardní implementace vyhledává soubory s rozložením (*layoutem*) relativně k souboru *presenteru*, který jsme v naší struktuře přesunuli a došlo by tak opět ke kritické chybě v aplikaci.

5.3.1 Integrace Nette a Doctrine

Integrace *frameworku* Nette a databázové vrstvy Doctrine se skládá z několika kroků. Jako první je potřeba registrovat adresář se zdrojovými soubory Doctrine do třídy `Doctrine\Common\ClassLoader`, která se stará o automatické načítání tříd souvisejících s touto databázovou vrstvou. To provedeme v souboru `app/bootstrap.php` vytvořením nové instance třídy `loaderu`, které zadáme jako první parametr jmenný prostor Doctrine a jako druhý parametr cestu k adresáři `libs`. Následuje vytvoření instance `EntityManageru`, který slouží jako bod přístupu k databázi. Aby bylo možné pohodlné ladění pomocí nástrojů Nette, budeme také potřebovat výpis prováděných dotazů do takzvaného *Debug Baru*. Protože se jedná o netriviální operace je pro tuto konfiguraci připraveno hotové řešení ve formě ukázkové aplikace [16]. Toto řešení zajistí nejen vytvoření `EntityManageru` a napojení na *Debug Bar*, ale přináší i podporu pro použití konzolových příkazů Doctrine.

5.4 Jádro systému

Jádro systému je složeno z několika součástí, které si v této části postupně popíšeme. První a stěžejní částí jádra je třída `Core`. Tato třída zajišťuje načtení všech požadovaných souborů důležitých pro běh aplikace a spravuje *hooky*. Třída také registruje a ruší moduly. Následně se podíváme na pomocnou třídu `Dictionary` spravující jazykové mutace aplikace. Nakonec si popíšeme vlastnosti a hierarchii jednotlivých tříd *presenterů*.

5.4.1 Návrhový vzor *singleton*

Na začátku této části bychom se měli zmínit o návrhovém vzoru *singleton*, protože hned dvě třídy jádra a to `Core` a `Dictionary`, tento návrhový vzor využívají. Implementace tohoto návrhového vzoru je značně závislá na použitém programovacím jazyce. V našem případě si tedy ukážeme implementaci v jazyce PHP.

Nejprve si řekněme, co nám použití tohoto návrhového vzoru přináší a jaký problém řeší. Důvodem pro použití vzoru *singleton* je požadavek na existenci pouze jedné instance dané třídy v rámci celé aplikace a zároveň globální dostupnost této instance. V našem případě jde například o třídu `Core`, kterou chceme mít dostupnou v rámci celé aplikace. Jádro (třída `Core`) však zajišťuje poměrně náročnou operaci načtení modulů a jejich *hooků*, kterou nechceme vykonávat opakovaně.

Nyní se můžeme podívat přímo na implementaci v jazyce PHP. Objekt se v PHP vytváří pomocí speciální metody `__contruct`, která je volána při požadavku na vytvoření instance dané třídy. Tato metoda může mít libovolný počet parametrů. V rámci třídy může být metoda definována jen jednou, ale nemusí být definována vůbec. Pro naši implementaci je důležité metodu

`__construct` definovat a to s modifikátorem přístupu `private`. Pokud bychom nechali naši třídu takto, nebylo by možné vytvořit žádnou její instanci. Budeme tedy pokračovat přidáním atributu `instance`, deklarovaného jako `private` a `static`. Právě tento atribut nám bude soužit pro udržování unikátní instance třídy. Pro získání instance si zavedeme metodu `getInstance`, která bude deklarována jako `public` a `static`. V těle metody se zkontroluje, zda je v atributu `instance` přítomna instance třídy nebo má-li nedefinovanou hodnotu (`null`). V případě že hodnota atributu není definovaná, zavolá se privátní metoda `__construct` a získaná instance se uloží do atributu `instance`. Výsledkem metody je pak vždy hodnota atributu `instance`. Pro názornost uvedeme ještě příklad implementace přímo na zdrojovém kódu [17].

```
class Singleton {
    private static $instance = null;
    private function __construct() {}
    public static function getInstance() {
        if (self::$instance === null) {
            self::$instance = new Singleton();
        }
        return self::$instance;
    }
}
```

5.4.2 Třída AbstractModule

Třídu od třídy `AbstractModul` musí dědit každá třída modulu (například pro modul správy uživatelů je to třída `UserModule`). Tato abstraktní třída obsahuje metodu, která vrací pole s názvy *hooků* pro tento modul. Právě tato metoda je volána při registraci *hooků* jádrem systému. O tom zda se jedná o běžnou metodu nebo *hook* se rozhoduje podle jména metody. *Hooky* musejí vždy začínat slovem `hook`. Aby nedocházelo ke kolizím jmen *hooků* mezi moduly, doporučuje se dodržovat konvence pojmenování `hookJmenoModuluNazevFunkce`, například `hookUserGetList`. Pro získání seznamu metod, které třída modulu obsahuje, se využívá reflexe, jak ukazuje následující útržek zdrojového kódu. Protože je třída `AbstractModule` použita pouze jako předek při vytváření nových tříd modulů, je nutné získat reflexi třídy, která metodu volá.

```
$reflection = new ReflectionClass(get_called_class());
$methods = $reflection->getMethods();
```

5.4.3 Třída Core

Třída `Core` implementuje návrhový vzor *singleton* a nabízí rozhraní pro registraci a spouštění *hooků*. Dále obsahuje odkaz na objekt `EntityManageru`, který zastřešuje práci s databázovou vrstvou. Instance této třídy je vytvořena ještě před startem vlastního *frameworku*. Po vytvoření instance následuje přiřazení instance `EntityManageru` do atributu `em`, prostřednictvím metody `setEm` a načtení modulů prostřednictvím metody `loadModules`.

Při načítání modulů se začíná otevřením adresáře `app/modules`. Následně se pro všechny obsažené adresáře provede operace pro načtení modulu do aplikace. Metoda vyhledá v databázi modul se jménem vytvořeným složením názvu adresáře a slova *Module* (například pro adresář `user` se bude hledat `UserModule`). Nenalezení záznamu v databázi značí, že se jedná o nový modul. V takovém případě je do systému přidán nový záznam a systém se pokusí nalézt soubor `create.sql`. Pokud je soubor v adresáři modulu přítomen, provede se jeho obsah jako SQL skript. Obsahem skriptu může být například vytvoření nových tabulek pro potřeby modulu, případně

rozšíření stávajících tabulek. Jako další krok se provede načtení souboru `config.ini`. Z tohoto souboru jsou načteny informace o verzi, autorovi modulu a podobně. Informace získané z konfiguračního souboru jsou uloženy do databáze v rámci záznamu o modulu. Nyní se ověří nastavení parametru `active`, který slouží pro aktivaci/deaktivaci modulu v rámci CMS. V případě, že je modul nastaven jako aktivní, následuje získání seznamu *hooků*, které tento modul implementuje. Pro získání *hooků* slouží metoda `getHooks`, kterou musí implementovat každý modul a jejíž funkčnost byla popsána v části 5.4.2. Získaný seznam je následně uložen do atributu `hooks`. Tento atribut je reprezentován asociativním polem, kde klíčem je název *hooku* a hodnotou je pole, obsahující seznam modulů, které daný *hook* implementují. Poslední akcí provedenou v rámci načtení modulu je vytvoření instance třídy modulu (například instance třídy `UserModule` pro modul `user`) a její uložení do atributu `instancesOfModules`. Instance jsou v tomto atributu opět uloženy v asociativním poli pod klíčem tvořeným názvem modulu.

Bezpečné volání funkcí a metod

Jak jsme si řekli dříve, *hooky* zavádíme z důvodů nízké vazby mezi moduly a zamezení fatálních chyb při absenci modulu. V předchozí části jsme si popsali jakým způsobem je ve třídě `Core` vytvořen seznam *hooků* a seznam instanci tříd modulů, které tyto *hooky* implementují. Nyní zbývá odpovědět na otázku, jakým způsobem můžeme *hooky* bezpečně spouštět, tak aby nedošlo k volání neexistující funkce nebo metody, které by způsobilo fatální chybu aplikace.

Odpovědí jsou vestavěné funkce PHP `call_user_func` a `call_user_func_array`. Obě funkce mají jako první parametr název metody nebo funkce, kterou se pokusí vykonat. Pokud se jedná pouze o název funkce nebo statické metody, můžeme parametr zadán jako text. V případě, že chceme volat metodu nad instancí jisté třídy, má parametr formu dvouprvkového pole s instancí třídy a názvem metody. Obě funkce se pak liší formou předávání parametrů pro volanou funkci/metodu. Výsledkem provedení funkce je buď výsledek po provedení požadované funkce/metody nebo logická hodnota `FALSE` a to v případě, že se požadovanou funkci/metodu nepodaří nelézt nebo spustit. Následuje několik příkladů použití funkcí `call_user_func` a `call_user_func_array`.

```
// volání funkce s jedním parametrem
call_user_func("funkce", param1);
// volání statické metody třídy Trida se dvěma parametry
call_user_func("Trida::funkce", param1, param2);
// volání metody pro instanci třídy Trida s jedním parametrem
call_user_func(array(new Trida, "funkce"), param1);
// volaní funkce s třemi parametry ve formě pole
call_user_func_array("funkce", array(param1, param2, param3));
```

Zpracování *hooku*

Nyní si popíšeme, jakým způsobem je volání *hooku* zpracováno v rámci vyvíjené aplikace. Pro spuštění *hooku* je určena metoda `runHook` třídy `Core`, která má následující parametry:

- `hook` – Název *hooku*, který má být proveden.
- `parameters` – (volitelný) Parametr nebo pole parametrů, které se předají volané funkci/metodě. Pokud bude parametr jeden nebo žádný, použije se pro provedení funkce/metody funkce `call_user_func`. V případě že na místě tohoto parametru bude pole, použije se funkce call_user_func_array.
- `moduleName` – (volitelný) Pokud bude uvedeno jméno modulu, provede se *hook* pouze pro tento modul a to i v případě, že *hook* implementuje více modulů.

Provedení hooku lze popsat následujícím algoritmem:

1. Volání metody `runHook` nad instancí třídy `Core`.
2. Z atributu `hooks` se získá seznam modulů, které požadovaný *hook* implementují.
3. Pokud byl v bodě 2 získán alespoň jeden název modulu, tak zpracování pokračuje na bodě 4, jinak zpracování končí a metoda vrací logickou hodnotu `FALSE`, značící že požadovaný *hook* nelze provést.
4. Postupně se pro všechny názvy modulů z kroku 2 získá instance třídy modulu z atributu `instancesOfModules`. Získaná instance je následně použita při volání požadovaného *hooku* prostřednictvím funkce `call_user_func` nebo `call_user_func_array`, podle hodnoty parametru `parameters`.
5. Výsledek provedení *hooku* nad každým modulem je uložen do asociativního pole s výsledky pod klíčem určeným jménem modulu.

V předchozím popisu zpracování *hooku* jsme pro zjednodušení zvažovali variantu, kdy není nastaven parametr `moduleName` funkce `runHook`. Pokud by parametr byl nastaven, došlo by po bodu 2 k ověření výskytu požadovaného modulu v seznamu z bodu 2. Následně by v bodě 4 došlo k volání pouze nad požadovaným modulem a v bodě 5 by byl výsledek vrácen bez zařazení do asociativního pole.

Příklad zpracování *hooku*

Pro lepší pochopení si nyní uveďme konkrétní příklad zpracování *hooku*. Uvažujme, že v systému máme dva moduly a to `UserModule` a `PageModule`. Oba moduly implementují hook `hookAddAdminMenu` (v souborech UserModule.php a PageModule.php je metoda `hookAddAdminMenu`). Po načtení modulů budou atributy instance třídy `Core` obsahovat následující:

```
hooks["hookAddAdminMenu"] = ["UserModule", "PageModule"]
instancesOfModules["UserModule"] = new UserModule
                  ["PageModule"] = new PageModule
```

Uvažujme následující volání metody `runHook("hookAddAdminMenu")`. V bodě 2 tedy získáme seznam `[UserModule, PageModule]`. Podmínka pro bod 3 je splněna, protože seznam obsahuje právě dva moduly. Provedení bodů 4 a 5 reprezentuje následující kód:

```
$hook = "hookAddAdminMenu";
$instUM = $this->instancesOfModules["UserModule"];
$instPM = $this->instancesOfModules["PageModule"];

vysledek["UserModule"] = call_user_func(array($instUM, $hook));
vysledek["PageModule"] = call_user_func(array($instPM, $hook));
```

5.4.4 Třída Dictionary

Třída `Dictionary` poskytuje metody pro zajištění multijazyčnosti administrační části systému. Stejně jako třída `Core` i tato je implementací návrhového vzoru *singleton*. Při vytvoření instance třídy je automaticky vybudován slovník pro celou aplikaci. Slovník je zde reprezentován pomocí atributu `dictionary` a má formát asociativního pole. Záznam ve slovníku má pak následující tvar `[jazyk][název modulu][index hesla][překlad v daném jazyce]`. Pro získání překladu pro jednotlivé jazyky a moduly se postupně prohledávají adresáře modulů, tak jako v případě načítání modulů jádrem systému. V každém adresáři modulu je následně prohledán adresář

`/languages`. Každý soubor, který odpovídá formátu popsanému v části 4.1.4 je otevřen a jeho obsah je dekódován jako formát JSON a přidán do slovníku. Index `jazyk` je dán názvem souboru, `název modulu` je získán z aktuálně zpracovávaného adresáře modulu a zbývající dvojice `index hesla` a jeho `překlad` už jsou tvořeny dekódovaným obsahem právě zpracovávaného souboru. Díky popsanému přístupu je přidávání nových jazykových mutací modulů snadné a spočívá v prostém přidání souboru do adresáře `/languages`.

Před začátkem používání slovníku je nutné nastavit pomocí metody `setLangs` výchozí a náhradní jazyk pro překlad. Náhradní jazyk se použije v případě, že by překlad hesla ve výchozím jazyce nebyl nalezen.

Překlad požadovaného hesla získáme prostřednictvím metody `translate`. Metoda má jeden parametr a to index požadovaného hesla ve tvaru `názevModulu_heslo` (například `user_password`). Výsledkem je pak překlad hesla ve výchozím nebo náhradním jazyce. Pokud není nalezen překlad ani pro jeden z nastavených jazyků, je vrácen text se zprávou o chybějícím překladu.

5.4.5 Hierarchie presenterů

Poslední částí, kterou probereme v souvislosti s jádrem je hierarchie *presenterů*. Jak jsme si řekli v části 4.1.1, je *presenter* třída, obsahující business logiku aplikace a řídící zpracování požadavku. Protože *presentery* administrační části mají některé společné požadavky například na zabezpečení a kontrolu oprávnění, je výhodné použít následující hierarchii *presenterů*.

Nette\Application\UI\Presenter

Jedná se o *presenter*, který je součástí Nette *frameworku*. Tato třída obsahuje implementaci množiny základních metod souvisejících s životním cyklem *presenteru*, nastavením *layoutu* a *templatu*, kontrolou zda se jedná o AJAX požadavek a podobně. Třída je použita jako základ pro odvozování uživatelských *presenterů*.

BasePresenter

Nejjednodušší *presenter*, který dědí od třídy `Nette\Application\UI\Presenter`. Obsahuje navíc atributy `core` a `em`. Atribut `core` nese instanci jádra systému. Atribut `em` obsahuje instanci třídy `Doctrine\ORM\EntityManager`, která slouží pro práci s databázovou vrstvou. Tento *presenter* navíc přepisuje metodu `formatLayoutTemplateFiles`, jak bylo popsáno v části 5.3.

Třída může být použita pro vytvoření *presenteru*, u kterého nejsou kladeny požadavky na přihlášení uživatele a kontrolu přístupových práv.

AdminPresenter

Třída dědí od třídy `BasePresenter`. Nejdůležitější součástí této třídy je přepsaná metoda `startup`. Tato metoda je součástí životního cyklu *presenteru* a vykoná se ještě před spuštěním zpracování konkrétní požadované akce. V rámci ní se postupně provedou tyto akce:

- načtení globální konfigurace pro CMS
- kontrola přihlášení uživatele
- načtení slovníku

Třída dále nabízí důležité metody, které využívají odvozené *presentery*. Následuje jejich výčet se stručným popisem:

- `isAllowed` – Metoda slouží pro ověření přístupových práv uživatele k požadovanému obsahu.
- `templatePrepareFilters` – Přepsaná metoda, která obsahuje přidání makra `dictionary` do šablonovacího systému Latte. Díky tomu je možné jednoduché použití slovníku v rámci šablon stránek.
- `translate` – Jedná se o nadstavbu nad stejnojmennou metodou třídy `Dictionary`. Metodě stačí předat heslo pro překlad bez názvu modulu. Název modulu je doplněn automaticky.

Od této třídy jsou odvozeny všechny *presentery* administrační části CMS.

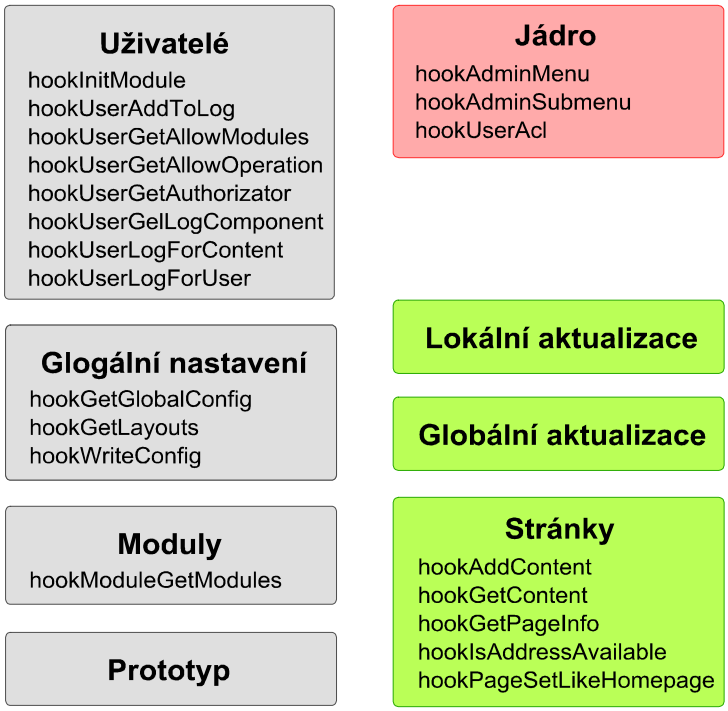
5.5 Moduly

Nyní si popíšeme nejdůležitější moduly CMS systému z pohledu jejich implementace a funkcionality. Samotné moduly lze rozdělit do dvou skupin a to moduly, systémové a rozšiřující. Systémové moduly by v systému měli být vždy přítomné, protože nabízejí základní funkcionalitu jako přihlašování uživatelů, správu modulů a podobně. Mezi systémové patří tyto moduly:

- modul pro správu uživatelů a práv
- modul globální konfigurace
- modul pro správu modulů
- vzorový modul

Vzorový modul je prototyp pro vývoj nových modulů a pro běh systému není důležitý. Jeho přítomnost v systému je však vyžadována ve specifikaci, a proto je mezi systémové moduly zařazen. Dále by bylo možné mezi systémové moduly zařadit modul pro správu stránek. Při jeho absenci nefunguje veřejná část webové prezentace. To však nemá vliv na funkčnost administrační části, a proto mezi systémovými moduly zařazen není.

Na následujícím obrázku (Obrázek 4) jsou znázorněny všechny moduly CMS s *hooky*, které implementují. Jádro není modulem, ale znázorňuje *hooky* potřebné pro registraci modulu do systému. *Hooky* uvedené u jádra zároveň implementují i všechny moduly. Systémové moduly jsou na obrázku odlišeny šedou barvou a uživatelské zelenou.



Obrázek 4: Moduly CMS

5.5.1 Vývoj a integrace nového modulu

Ještě před samotným popisem již implementovaných modulů si ukažme, jakým způsobem vytvořit nový modul a jak ho integrovat do CMS. Pro tvorbu nových modulů je součástí systému vzorový modul ve složce `app/modules`. Modul nese název `prototype` a obsahuje následující soubory a adresáře:

`/languages`	složka určená pro soubory s překlady
`/cs.lang`	ukázkový jazykový soubor pro češtinu s několika hesly
`/models`	složka připravená pro Doctrine entity
`/presenters`	složka určená pro uživatelské *presentery*
`/PrototypePresenter.php`	ukázkový *presenter* s jednou akcí a ukázkou nastavení titulku stránky
`/templates`	složka pro šablony
`/Prototype`	šablony *presenteru* `PrototypePresenter`
`/default.latte`	šablona pro akci `default`
`/PrototypeModule.php`	soubor modulu s ukázkovou implementací *hooků* `hookAdminMenu`, `hookAdminSubmenu` a `hookUserGetAcl`
`/config.ini`	konfigurační soubor modulu

Při vytváření nového modulu tedy začneme zkopírování a přejmenováním tohoto vzorového modulu. Následně přejmenujeme soubory *presenteru*, modulu a adresář se šablonami tak, že název `Prototype` zaměníme za námi požadované jméno nového modulu. V souboru `config.ini` nalezneme řádek `system = TRUE`, který značí, že se jedná o systémový modul. Pro zrušení této volby stačí řádek vymazat. Když v tuto chvíli otevřeme administrační část CMS a přejdeme do sekce správy modulů, bude již nový modul v seznamu.

Systém nabízí *hooky*, které nám umožní hlubší integraci modulu se systémem. Implementace všech popisovaných *hooků* je uvedena v komentářích v souboru `PrototypeModule.php`.

- `hookAdminMenu` – Registrace položky do hlavního administrátorského menu.
- `hookAdminSubmenu` – Registrace položek do submenu daného modulu.
- `hookUserGetAcl` – Pomocí tohoto *hooku* je možné do systému přidat zdroje a oprávnění pro řízení přístupu. Správa řízení přístupu je součástí modulu pro správu uživatelů a budeme se jí věnovat později.

5.5.2 Modul pro správu uživatelů

Jak bylo uvedeno, jedná se o systémový modul. Modul zajišťuje správu uživatelů a uživatelských skupin, přihlašování uživatelů do CMS, řízení přístupu ke zdrojům systému a logování činnosti uživatele.

Správa uživatelů a uživatelských skupin

V této části stojí z pohledu implementace za povšimnutí pouze zabezpečení uživatelského hesla pomocí techniky *salted hash* [18]. Technika se skládá ze dvou kroků. Jako první vezmeme heslo v otevřené podobě a na libovolné místo k němu připojíme takzvaný *salt*. Jako *salt* je označován libovolný řetězec, kterým heslo takzvaně osolíme. Následně na vzniklý řetězec použijeme hashovaní funkci. Tímto se vyhneme útoku, kdy má útočník databázi *hash* kódů pro nejčastěji používaná hesla a pokouší se provést slovníkový útok. Abychom situaci útočníkovi zkomplikovali ještě více, je dobré použít odlišného *salt* pro každé heslo. Toto vylepšení nám zaručí, že výsledný *hash* kód bude rozdílný i pro stejná hesla. V naší aplikaci používáme jako salt první tři písmena uživatelského jména připojené na začátek hesla a *hashování* pomocí funkce SHA1.

Přihlašování uživatelů

Protože je tento proces přímo součástí použitého *frameworku*, muselo dojít k těsnému propojení jádra aplikace a modulu. Pro přihlašování je použita třída `Authenicator`, kterou nalezneme v adresáři s modely. Tato třída implementuje rozhraní `Nette\Security\IAuthenticator` a do *frameworku* je zaregistrována v konfiguračním souboru `config.neon`. Rozhraní `Nette\Security\IAuthenticator` obsahuje metodu `authenticate`, kterou musíme ve třídě `Authenticator` naimplementovat. V této metodě je provedeno ověření totožnosti uživatele a vytvoření objektu `Nette\Security\Identity`, který nese informace o přihlášeném uživateli a je dostupný v rámci celé aplikace. Pokud ověření totožnosti uživatele selže, je vyvolána výjimka `Nette\Security\AuthenticationException`.

Řízení přístupu

Za nejzákladnější kontrolu pro přístup ke zdroji můžeme považovat skutečnost, že je daný uživatel přihlášen. Pro zjištění stavu přihlášení uživatele, nabízí třída `Nette\Security\Identity` metodu `isLoggedIn`. Takový přístup k řízení přístupu je však nedostatečný a proto musíme zavést další prostředky, abychom umožnili řízení přístupu detailněji. Nette nám zde opět vychází vstříc a to třídou `Nette\Security\Permission`. Tato třídu operuje s následujícími elementy:

- role – vyjadřuje roli uživatele v systému
- zdroj – představuje logickou část systému (například *presenter*)
- operace – jedná se o nějakou akci provedenou uživatelem (například akce v *presenteru*)

Třída `Nette\Security\Permission` obsahuje řadu metod pro správu těchto elementů a definování vztahů mezi nimi. Pro nás jsou nejzajímavější tyto metody:

- `addRole` – přidání role (je možné vytvořit i hierarchii rolí)
- `addResource` – přidáni zdroje
- `allow` – povolení přístupu ke zdroji pomocí deklarovaných operací
- `deny` – zamezení přístupu ke zdroji pomocí deklarovaných operací
- `isAllowed` – ověření přístupu uživatele ke zdroji za pomoci zadané operace

Popsali jsme si, jaké prostředky nám nabízí použitý *framework* a nyní se budeme věnovat konkrétní implementaci řízení přístupu v naší aplikaci. Ověření, zda je uživatel přihlášen, je provedeno v metodě `starup` v `AdminPresenteru`. Pro řízení přístupu jsou jako role použity uživatelské skupiny. Zdroje a operace si registruje každý modul samostatně pomocí implementace hooku `hookUserGetAcl`. Abychom umožnili uživateli nastavovat pravidla pro řízení přístupu v CMS je nutné zdroje a operace uložit do databáze. Protože se jedná o poměrně náročnou operaci, kdy je potřeba porovnat aktuální obsah databáze se seznamem získaným voláním hooku `hookUserGetAcl`, zavedeme si *cachování*. *Cachování* spočívá v uložení získaného seznamu zdrojů a operací do souboru. Při dalším dotazu je obsah souboru porovnán s aktuálně získaným seznamem a jsou-li stejné, není potřeba databázi měnit. Díky tomu se vyhneme náročnému ověřování, zda zdroje a operace v databázi již existují.

Povolení operace pro uživatelskou skupinu se děje prostřednictvím uživatelského prostředí administrační části systému. V databázi se fakt, že má daná skupina právo na některou z operací, projeví vytvořením záznamu s identifikátorem příslušné skupiny a operace. Z toho vyplývá, že pro řízení přístupu se používá přístup „Co není povoleno je zakázáno“.

K vlastnímu ověření zda má uživatel právo přístupu k danému zdroji za pomoci dané akce, slouží metoda `isAllowed` ze třídy `AdminPresenter`. Protože je `AdminPresenter` předkem všech *presenterů* administrační části webu, je i tato metoda dostupná v každém z nich. Při zamítnutí přístupu je uživatel o tomto faktu informován chybovou zprávou a je přesměrován na obrazovku pro přihlášení. [19]

Logování aktivity uživatele

Obecně v rámci CMS pracujeme s obsahem, který se často mění. Za změny obsahu nesou zodpovědnost uživatelé. Aby bylo možné dokázat, kdo za provedené změny nese zodpovědnost, nabízí systém možnost logování aktivity uživatele. Pro tento účel jsou určeny následující *hooky* implementované ve třídě `UserModule`:

- `hookUserAddToLog` – přidání nového záznamu do logu
- `hookUserLogForUser` – získání záznamu aktivity pro vybraného uživatele
- `hookUSerLogForContent` – získání záznamu změn pro vybraný obsah
- `hookUserGetLogComponent` – vrací zformátovaný záznam změn pro požadovaný obsah v podobě komponenty Gridito

Protože *hook* pro přidání záznamu do logu požaduje poměrně značný počet složitě strukturovaných parametrů, nabízí třída `AdminPresenter` metodu `addToLog`, která *hook* `hookUserAddToLog` obaluje a doplňuje některé parametry automaticky podle aktuálního kontextu (aktuálně přihlášeného uživatele, předpony modulu a podobně).

Implementace logování přinesla zajímavý problém a to jak se vypořádat s odkazem na různý typ obsahu. Představme si situaci, kdy chceme zaznamenávat změny v obsahu webové stránky. U položky logu v databázi tedy budeme mít sloupec s cizím klíčem odkazujícím na upravenou stránku. Pokud přidáme do logu sledování změn v osobních údajích uživatele, budeme muset přidat další sloupec s cizím klíčem vedoucím na uživatele. S každým novým obsahem by nám tak vznikal nový sloupec v tabulce logu. Zároveň bychom přišli o nezávislost modulů, protože by vznikl požadavek na změnu tabulky, která nenáleží danému modulu. Abychom tento problém vyřešili, nebudeme odkazovat na obsah pomocí cizích klíčů v tabulce, ale uložíme do databáze dvojici jméno entity a id entity. Právě tyto dva parametry potřebuje metoda `find` z použité databázové vrstvy Doctrine pro nalezení konkrétní entity.

5.5.3 Modul globální konfigurace

Modul slouží pro globální nastavení administrační části webové prezentace. Jednou z voleb je nastavení souboru s rozložením stránky (*layoutem*) pro administraci a pro přihlašovací obrazovku. Dostupné *layouty* získává systém automaticky z adresáře `app/templates`. Název souboru s *layoutem* musí v Nette začínat znakem `@`. Aby bylo možné rozlišit layouty určené pro veřejnou a administrační část, musí *layout* pro administrační část začínat dvojicí symbolů `@_`. Další volbou je kaskádový styl určující vzhled administrace, jejichž seznam je generován systémem z adresáře `www/CMS/css`. Poslední dostupnou volbou je pak výchozí jazyk pro administraci. Dostupné jazyky je možno získat z instance třídy `Dictionary` z atributu `availableLangs`, který obsahuje všechny jazyky nalezené ve fázi výstavby slovníku. Výchozí jazyk nastavený v rámci globální konfigurace je použit při nastavení jazyků pro třídu `Dictionary` při přihlášení uživatele. Jako výchozí je použit jazyk nastavený uživatelem v rámci svých osobních údajů a jako náhradní jazyk je použit právě jazyk z globální konfigurace.

Veškeré nastavené hodnoty nejsou ukládány do databáze, ale pro jejich uložení se používá konfigurační soubor `config.ini`. Tento soubor se nachází v adresáři `app/modules`.

Modul dále nabízí několik *hooků*, které mohou využít ostatní moduly. Následuje jejich seznam a stručný popis jejich funkcionality:

- `hookGetLayouts` – Vrátí seznam všech souborů s *layoutem*. *Hook* má jeden nepovinný parametr typu `boolean`. Parametr určuje, zda chceme získat seznam *layoutů* pro veřejnou nebo administrační část systému.
- `hookGetStylesheets` – Výsledkem je seznam souborů s příponou `css` pro zadaný adresáš.
- `hookWriteConfig` – Umožňuje zapsat do konfiguračního souboru. Data jsou očekávána ve formě asociativního pole.

5.5.4 Modul pro správu modulů

Modul umožňuje aktivovat/deaktivovat jednotlivé moduly. Pokud je do systému přidán nový modul, je automaticky nastaven jako neaktivní. Toto nastavení způsobí, že modul není dostupný z menu administrace a nejsou dostupné ani jeho *hooky*. Moduly je možné libovolně aktivovat a deaktivovat podle potřeby. Výjimku tvoří systémové moduly, které jsou v rámci výpisu odlišené a které jsou automaticky aktivovány ve fázi nahrání do systému. Systémové moduly neleze deaktivovat. Pozice modulu ve výpisu všech modulů odráží pořadí modulu v administračním menu. Pro změnu pořadí slouží šipky na řádku u každého modulu. Nový modul je automaticky zařazen na poslední pozici.

V této části bychom se měli zmínit o komponentě Gridito. Jedná se o komponentu pro Nette, která umožňuje výpis tabulkových informací. Součástí implementace je i podpora pro stránkování a definování uživatelských akcí u jednotlivých položek výpisu. Gridito je stylované pomocí JQuery UI a umožňuje tedy snadnou změnu vzhledu. Důležitou vlastností je podpora pro databázovou vrstvu Doctrine, kterou využíváme v našem projektu.

Důvod proč si Gridito zmiňujeme právě u tohoto modulu, je odhalený problém s použitím textového primárního klíče pro výběr dat. Ve třídě `Gridito\DoctrineQueryBuilderModel` se totiž v metodě `getItemByUniqueId` používá explicitní přetypování primárního klíče na typ `integer`. My však používáme v databázi jako primární klíč pro modul jeho jméno. Bylo tedy potřeba vytvořit novou třídu, dědící od `Gridito\DoctrineQueryBuilderModel` a přepsat metodu `getItemByUniqueId`, tak aby respektovala i primární klíč typu `string`.

5.5.5 Modul pro správu stránek

Modul pro správu stránek obstarává veškerou základní funkcionalitu veřejné části systému a je to první nesystémový modul. V rámci popisu tohoto modulu se postupně zaměříme na úpravu webových stránek, globální nastavení webové prezentace a zpřístupnění veřejně dostupné stránky přes zadanou adresu.

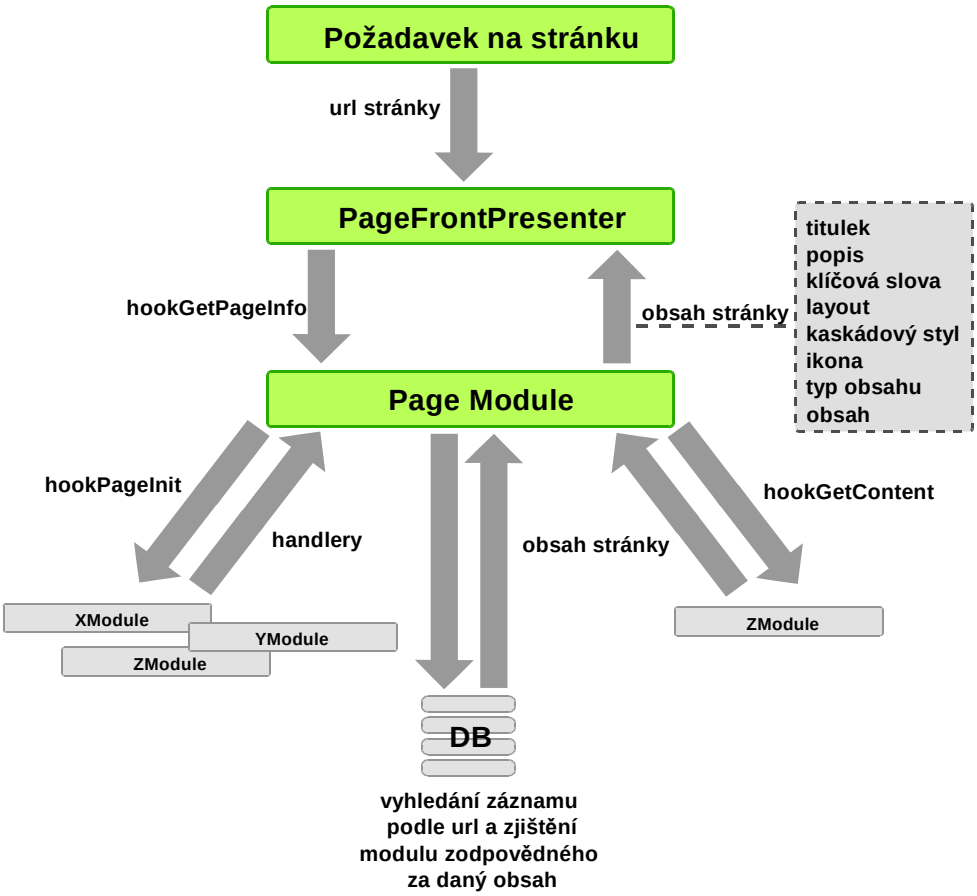
Začneme popisem základní funkcionality modulu. Modul umožňuje přímo ve výpisu stránek tuto stránku publikovat nebo jí skrýt a nastavit stránku jako domovskou (*homepage*). Jednotlivé stránky je možné editovat. Součástí editace jsou položky týkající se SEO optimalizace jako popis, klíčová slova a titulek stránky. Dále lze samostatně pro každou stránku nastavit *layout*. Možné *layouty* jsou získány pomocí hooku `hookGetLayouts`, který implementuje modul globální konfigurace. Položka adresa určuje adresu stránky na veřejné části webu a musí být v rámci systému jedinečná. Aby bylo možné unikátnost adresy kontrolovat i z dalších modulů, nabízí modul stránek *hook* `hookIsAddressAvailable`. Pro úpravu vlastního obsahu stránky je uživateli dostupný WYSIWYG editor markItUpEditor, který využívá značkovací jazyk Textile. Jedná se o jednoduchý značkovací jazyk, jehož syntax lze nalézt například na [20]. Uživatel vkládá značky pomocí menu použitého editoru, tak jak je běžné u jiných WYSIWYG editorů. Pouze pro ilustraci si uveďme příklady několika značek:

```
h1. nadpis první úrovně
* položka nečíslovaného seznamu
__kurzíva__
*tučný text*
```

Další sekcí modulu stránek je nastavení. Zde najdeme nastavení týkající se celé webové prezentace. Lze nastavit položky pro SEO optimalizaci stejně jako u stránek. Toto nastavení se využije v případě, že bude zobrazena stránka, která nebude mít jednu z položek (titulek, popis nebo klíčová slova) vyplněnou. Další volbou je výchozí *layout* a kaskádový styl. Pomocí vybraného kaskádového stylu je pak možné snadno měnit vzhled celé výsledné webové prezentace. Nakonec zde nalezneme ještě textové pole pro zadání cesty k ikoně stránek.

Nyní se podívejme na to, jakým způsobem je vytvořena stránka pro veřejnou část webu. Pro demonstraci je přiložen Obrázek 5. Všechny požadavky na veřejnou část webu jsou směrovány na akci `default` *presenteru* `PageFrontPresenter`. Jako parametr je akci předána požadovaná url adresa. Například při požadavku `www.domena.cz/clanek/1` je v parametru `url` předáno `clanek/1`. `PageFrontPresenter` spustí *hook* `hookGetPageInfo` nad modulem `PageModule`. V těle *hooku* jsou mimo jiné provedeny následující tři akce:

- `hookPageInit` – Zavolá se nad všemi moduly. Umožňuje zaregistrovat *handlery* (*hooky*), které má modul vykonat před a po získání obsahu stránky. *Hooky*, které mají být vykonány před získáním obsahu, dostanou jako parametr url požadované stránky a mohou ho libovolně upravit. Tohoto může využít například modul pro správu menu. *Hooky*, které mají být vykonány po získání obsahu, dostanou jako parametr obsah stránky. Zde si jako příklad použití můžeme uvést modul pro obrázky, který v textu nahradí značky pro vložení obrázku skutečnou cestou k obrázku.
- Nalezení záznamu v tabulce `content`, podle požadované url. Ze záznamu získáme název modulu a parametr v podobě id pro zavolání *hooku* `hookGetContent`.
- `hookGetContent` – Zavolá se pouze nad modulem získaným ze záznamu z databáze. Jako parametr dostane id obsahu. Výsledkem volání tohoto *hooku* je asociativní pole s informacemi o požadované stránce. V modulu pro správu stránek se v případě potřeby doplní chybějící informace jako například titulek stránky nebo klíčová slova. Ty jsou získány z nastavení webové prezentace.



Obrázek 5: zpracování požadavku na veřejnou stránku

5.5.6 Modul pro lokální aktualizaci systému

Modul se skládá ze dvou částí. První z nich je správa nastavení aktualizace. Pro správné fungování aktualizace je požadováno několik základních nastavení:

- nastavení SVN serveru – Obsahuje nastavení adresy serveru, složky v které se požadovaná verze CMS nachází, přihlašovací jméno a heslo. Při validaci zadaných údajů se provede testovací připojení k SVN serveru.
- adresář pro zálohu – Relativní cesta k adresáři pro zálohu. Cesta je vztažena k aplikačnímu adresáři `app`. Do rodičovského adresáře je nutné vygenerovat soubor netterobots.txt, který obsahuje pravidlo `Disallow: /bckup`. Tímto zamezíme, aby třída pro automatické načítání tříd *frameworku* prohledávala adresář se zálohou. Pokud bychom tento krok neučinili, došlo by ke kolizi jmen tříd mezi zálohovanými soubory a soubory aktuální verze.
- výběr modulů – Umožňuje uživateli zvolit, které moduly mají být aktualizovány. Jádro CMS je aktualizováno vždy nezávisle na vybraných modulech.

Druhou částí modulu je samotná aktualizace. Tato část obsahuje také seznam záloh po dříve provedených aktualizacích s možností návratu k libovolné předchozí verzi. Samotná aktualizace je pak rozdělena do pěti fází. Každá z nich je volána samostatně prostřednictvím technologie AJAX, aby byl uživatel průběžně informován o aktuálním stavu zpracování. Nyní si jednotlivé fáze popíšeme.

Při načtení stránky se do `Session` uloží nastavení aktualizace. Po spuštění aktualizace je provedena inicializace, která spočívá v kontrole nastavení a vytvoření adresáře pro zálohu stávajícího systému. Jméno adresáře je odvozeno z aktuálního data a času. Absolutní cesta k tomuto adresáři je uložena do `Session`. Protože budeme na adresář v rámci jednotlivých fází aktualizace ještě několikrát odkazovat, ukažme si teď jeho obsah po provedení úspěšné aktualizace:

`/2012-07-12_10-47-40`	
`/core`	záloha jádra systému
`/libs`	třídy jádra
`/www`	soubory stylů, *javascriptové* knihovny, obrázky a podobné
`/database_dump`	
`/dump.sql`	SQL skript pro obnovení databáze
`/modules`	záloha modulů

Po vytvoření adresáře pro zálohu se provede záloha databáze. Ta spočívá ve vytvoření SQL skriptu pro obnovu databáze. Skript je pojmenován `dump.sql` a je uložen do adresáře `database_dump`. K vytvoření skriptu se používá knihovna Eesy MySQL Dump [21]. Knihovna však počítá s použitím nativních funkcí PHP pro práci s MySQL databází. Z tohoto důvodu bylo nutné provést několik zásadních změn, tak aby jí bylo možné integrovat databázovou vrstvou Doctrine.

Třetí fází aktualizace je uložení záznamu o bodu návratu do databáze. Záznam obsahuje seznam aktualizovaných modulů, absolutní cestu k adresáři se zálohou a nastavení SVN serveru při spuštění aktualizace. Protože byla záloha databáze provedena v předcházející fázi, nebude tento záznam po případné obnově dostupný.

Následuje záloha souboru aktuální verze. Postupně se zkopíruje obsah adresářů `libs/CMS`, `www/CMS` a vybraných modulů do odpovídajících umístění v adresáři pro zálohu. Obsah prvních dvou adresářů představuje soubory jádra. Zálohy modulů jsou umístěny v adresářích pojmenovaných podle názvu odpovídajícího modulu.

Poslední fází aktualizace je smazání původních souborů a jejich nahrazení novou verzí. Nová verze systému musí být stažena z SVN serveru. Pro tuto činnost použijeme knihovnu phpsvnclient, kterou musíme pro integraci s Nette opět modifikovat. [22] Knihovna je programována pouze v PHP a nemá žádné další závislosti. Díky tomu nebudou kladeny dodatečné nároky na *hosting* pro nasazení vyvíjeného CMS.

Zotavení po chybě a návrat ke starší verzi

Proces aktualizace je rozsáhlá operace a v jejím průběhu hrozí výskyt chyb, který by mohl způsobit nefunkčnost aplikace. Z tohoto důvodu je systém vybaven metodou `handleRollback` pro zotavení po chybě. Tato metoda se v závislosti na fázi, v jaké se aktualizace nachází, postará o navrácení aplikace do stavu před spuštěním aktualizace.

Dále je možné provést návrat na libovolnou předchozí verzi systému. Při tomto procesu dojde k nahrazení všech souborů jádra a modulů, které byly v rámci dané verze měněny. Zároveň se provede návrat stavu databáze na stav platný v době používání požadované verze. Protože se jedná o poměrně zásadní operaci, je uživatel před samotným spuštěním akce vyzván k potvrzení akce prostřednictvím modálního okna.

5.5.7 Modul pro globální aktualizaci

Jak jsme si popsali v kapitole 4.2, implementace aplikace pro hromadnou obsluhu bude spočívat v rozšíření modulu pro lokální aktualizaci o veřejně dostupné rozhraní (API) a vytvoření nového modulu, který bude udržovat aktualizované domény a šablony pro aktualizaci.

Pro komunikaci mezi těmito dvěma moduly, použijeme technologii vzdáleného volání procedur RPC (*Remote Procedure Call*). K tomuto účelu nám poslouží knihovna JSON RPC Server [23]. Jak je z názvu patrné, používá metoda pro komunikaci formát JSON, který má výrazně úspornější zápis oproti standardně používanému značkovacímu jazyku XML. Díky tomu je možné mezi RPC serverem a klientem přenášet menší objem dat při každém požadavku.

Serverová část (modul pro lokální aktualizaci)

RPC server je aplikace, která má některé své služby dostupné prostřednictvím veřejného rozhraní. V našem případě se jedná o aktualizované domény. Veřejné rozhraní je tvořeno skrze presenter `RemotePresenter` v modulu pro lokální aktualizaci. Tento presenter dědí pouze od třídy `BasePresenter` z čehož plyne, že je volně dostupný bez potřeby přihlášení. Pro účely autentizace je určen sdílený klíč. Volání požadované funkce se děje přes přístup na výchozí akci tohoto presentru (například `www.domena.cz/admin/update/remote`). Název volané funkce a parametry, jsou předány jako součást zprávy ve formátu JSON. Registrace nové funkce, probíhá v metodě `actionDefault` a to pomocí takzvaně magické funkce PHP `__set`. Pokud tedy máme instanci třídy RPC serveru, stačí pro registraci funkce `novaFunkce`, zapsat `$server-> novaFunkce = array($this, " novaFunkce");`. Všechny funkce mají stejný formát odpovědi a to dvouprvkové asociativní pole kde pod klíčem `status` je vrácena logická pravdivostí hodnota, podle výsledku operace a pod klíčem `msg` je upřesňující zpráva. Presenter nabízí pro potřeby vzdáleného volání následující funkce:

- `update` – Funkce má stejný efekt a využívá stejnou funkcionalitu jako spuštění lokální aktualizace na dané doméně a to včetně vytvoření zálohy a záznamu o aktualizaci v lokální databázi. Jediný rozdílem je, že záznam o aktualizaci obsahuje příznak signalizující vzdálené spuštění aktualizace.

- `rollback` – Slouží pro zrušení právě probíhající aktualizace při chybě a návrat systému do stavu před aktualizací.
- changeSharedKey – Vzdálená změna sdíleného klíče, který je používat pro autentizaci a šifrování parametrů, přenášených v rámci RPC zpráv.
- `getModules` – Získání seznamu modulů, které jsou na dané doméně přítomné.
- `revert` – Návrat na předchozí verzi systému. Využití funkce spočívá především v řešení nefunkčnosti systému po provedené aktualizaci.

Klientská část (modul pro hromadnou aktualizaci)

Modul pro hromadnou aktualizaci představuje v kontextu naší implementace RPC část klientskou. Využívá tedy vzdálené volání procedur implementovaných na jednotlivých serverech. Protože parametry použité při volání funkcí obsahují i citlivé údaje jako přístupové údaje na SVN server, jsou tyto parametry šifrovány. Šifrování probíhá pomocí vestavěné funkce PHP `mcrypt_encrypt` za použití šifrovacího algoritmu AES s 256bitovým klíčem v bokovém módu CBC. Jako klíč je použit MD5 *hash* získaný ze sdíleného klíče. Tento *hash* je po dalším použití algoritmu MD5 použit jako inicializační vektor pro šifrování.

Samotný modul se pak skládá ze dvou částí a to správy domén a správy aktualizačních šablon. Správa domén umožňuje kromě CRUD operací ještě vzdálenou změnu sdíleného klíče a návrat k předchozí verzi. Tyto funkce využívají RPC funkce serveru `changeSharedKey` respektive `revert`. U každé domény je uvedena url adresa, která slouží právě pro volání vzdálených procedur. Součástí informací o doméně je i přiřazená aktualizační šablona.

Část pro správu šablon nabízí základní operace nad těmito entitami. V rámci úpravy šablony je potřeba mít seznam modulů, které se mají aktualizovat. Ten získáme prostřednictvím volání vzdálené funkce `getModules` nad všemi doménami v šabloně. Výsledkem je pak sjednocení všech návratových hodnot. Samotná aktualizace se skládá z postupného volání funkce `update` pro všechny domény zařazené v šabloně. Jako parametry pro tuto operaci jsou předány přístupové údaje pro SVN server obsahující zdrojové soubory nové verze systému a seznam modulů, které mají být aktualizovány. V případě výskytu chyby při aktualizace na dané doméně, je na pozadí volána procedura `rollback`, která zaručí návrat systému do stavu před začátkem aktualizace. O případném neúspěchu aktualizace je uživatel informován chybovou zprávou a je mu nabídnuta možnost, spustit aktualizaci domény znovu.

6 Testování a ladění

Tato kapitola je věnována důležité fázi vývoje software a to testování. Protože s testováním přímo souvisí i nástroje pro ladění software při vývoji, zmíníme se krátce i o nich. Postupně se podíváme na prostředky, které nám nabízí použitý *framework* Nette, podporu pro jednotkové testování (*Unit testing*) v PHP a ladění (*deubbugování*) programu. V rámci této kapitoly budeme používat anglické názvy, které jsou ustálenější než jejich české ekvivalenty.

6.1 Nástroje v Nette

Součástí distribuce Nette *frameworku* je třída `Nette\Diagnostics\Debugger` [24], kterou uživatelé *frameworku* znají pod názvem Laděnka. Aby bylo možné služby této třídy využívat při vývoji a případně i provozu naší aplikace, je nutné jí zaregistrovat. To provedeme pomocí statické metody `enable` v souboru `app/bootstrap.exe`.

Jednou z výhod použití tohoto nástroje je vizuálně přehledné zobrazení chyb a výjimek s řadou doplňujících informací o stavu aplikace v době vzniku chyby. Jedná se o náhradu za nepřehledné chybové zprávy samotného PHP. V rámci výpisu je zobrazeny tyto informace:

- chybová zpráva
- část zdrojového souboru s označeným řádkem, na kterém došlo k chybě
- metody, jejichž volání vedlo k chybě (*call stack*) s možností zobrazení části dotčeného zdrojového kódu
- stav globálních proměnných prostředí v době chyby
- http požadavek s tabulkovým výpisem hlavičkových informací a obsah globálních proměnných `$_GET`, `$_POST` a `$_COOKIE`
- http odpověď

Dalším prostředkem sloužícím pro ladění programu je *Debug Bar*. Jedná se o plovoucí lištu zobrazenou na každé stránce, která obsahuje informace jako počet databázových dotazů s možností jejich zobrazení, dobu potřebná pro načtení stránky, alokovanou paměť a další. Lišta je rozšiřitelná a umožňuje tak uživateli přidání dalších důležitých informací.

Zatím jsme si popsali prostředky sloužící vývojáři v době implementace a testování software. Laděnka však nabízí i možnost logování chyb v době provozování systému. To, že je systém v ostrém provozu, je signalizováno buď automaticky pomocí detekce prostředí, nebo manuálně nastavením produkčního režimu. V tomto režimu jsou výpisy všech ladících informací do webového prohlížeče potlačeny a jsou místo toho směřovány do složky určené pro logování.

6.2 Unit testing

Unit testing [25], je metoda určená pro testování funkčních jednotek. PHP je jazyk objektový, takže si jako funkční jednotku můžeme představit například třídu nebo metodu. Testovaná jednotka by pak měla být do maximální možné míry izolována od svého okolí, tak aby výsledek testu byl na jejím okolí nezávislý. Protože s narůstajícím rozsahem aplikace poroste i počet potřebných testů, používají se pro hromadné spouštění a správu testů testovací *frameworky*. Mezi nejoblíbenější patří *framework* PHPUnit.

Vytvoření testu pro PHPUnit spočívá ve vytvoření třídy dědící od `PHPUnit_Framework_TestCase`. Název nové třídy musí navíc obsahovat příponu `Test` (například `mujTest`). Součástí třídy `PHPUnit_Framework_TestCase` je metoda `setUp`, kterou je v testovací třídě možno přepsat a která slouží k nastavení prostředí před samotným testováním. Názvy metod představující jednotlivé testy musejí začínat slovem `test` (například `testProgram`). Pro ověření správnosti výstupních hodnot se v rámci jednotlivých testů používají *asserty*, které jsou součástí PHPUnit.

V naší aplikaci je potřeba testovat třídy entit databázové vrstvy a *presentery*. Testování tříd entit sebou nese potřebu připojení k databázi. To získáme prostřednictvím vytvoření instance třídy `Doctrine\ORM\EntityManager`, buď v rámci jednotlivých testů, nebo v metodě `setUp`. Testování *presentrů* není již tak triviální záležitostí a skládá se z následujících kroků:

1. Vytvoříme si instanci *presentru* tak, že do konstruktoru příslušného *presentru* vložíme systémový kontejner, získaný ze statické třídy `Nette\Environment`.
2. Vytvoříme si požadavek na stránku jako instanci třídy `Nette\Application\Request`.
3. Zavoláme metodu `run` nad *presenterem* z kroku 1 a jako parametr jí předáme požadavek z kroku 2.
4. Jako výsledek metody `run` dostaneme instanci třídy implementující rozhraní `Nette\Application\IResponse`.

6.3 Ladící nástroje

Při implementaci jakéhokoli programu se setkáme s problémem opravy implementačních chyb. V případě, že potřebujeme chybu lokalizovat a zjistit co ji zapříčinilo, potřebujeme sofistikovanější techniku než výpis pomocí vestavěných funkcí PHP `echo` nebo `var_dump`. Tímto řešením může být například rozšíření pro PHP s názvem Xdebug. Instalace spočívá pouze ve vložení souboru Xdubug do složky určené pro rozšíření PHP a přidání několik řádků do konfiguračního souboru `php.ini`.

Toto rozšíření lze integrovat s vývojovým prostředím v našem případě s NetBeans. Tím získáme grafický ladící nástroj s možností vkládat zarážky, krokovat program, sledovat stav proměnných a tak dále. Právě sledování stavu proměnných a to včetně možnosti procházet pole nebo atributy objektů, je náhradou za zmiňované funkce `echo` a `var_dump`. Stav proměnné můžeme sledovat interaktivně během procházení programu po jednotlivých příkazech. Díky tomu je možné dopátrat se i jinak těžko odhalitelných chyb jako vedlejší efekty některých použitých funkcí nebo překrývání proměnných.

7 Závěr

Cílem práce bylo naimplementovat systém pro správu webové prezentace s použitím vhodného aplikačního rámce. Systém by také měl umožnit pohodlné šíření změn na více instalací najednou, aby bylo možné ho použít i v profesionální sféře.

Výsledný systém splňuje požadavky definované v části analýzy požadavků s výjimkou dvou rozšiřujících modulů pro správu obrázků a menu. Velká pozornost byla věnována především kvalitnímu návrhu a implementaci jádra systému. Návrh jádra vychází z použitého aplikačního rámce Nette, do kterého byla integrována databázová vrstva Doctrine. Výsledkem spojení těchto dvou projektů je jádro dobře udržovatelné, testovatelné a přináší kvalitní základ pro vývoj dalších rozšíření. Aby bylo možné tato rozšíření do systému pohodlně zapojovat, implementuje jádro koncept *hooků*, jehož účelem je snížení vazeb mezi rozšiřujícími moduly a jádrem systému.

Účelem vyvíjeného systému je tvorba a správa webové prezentace. Protože chceme, aby i rozšiřující moduly mohli přidávat svůj obsah do veřejné části webové prezentace, používá modul pro stránky rozdělení obsahu webové stránky a její obálky. Díky tomu není veřejná část odkázána pouze na obsah vytvořený v modulu pro tvorbu stránek, ale obsah stránky může dodat kterýkoli rozšiřující modul. To umožňuje například implementaci modulu pro internetový obchod, který bude jako obsah stránky vkládat vygenerované produktové stránky a podobně.

Hlavní výhodou systému jsou nabízené možnosti aktualizace. První z nich je určena pro lokální použití. Tento způsob aktualizace bude využit v případě, že je spravována pouze jedna instalace CMS. Aktualizace probíhá zcela automaticky a její součástí je i vytvoření zálohy stávajícího systému včetně databáze. Druhá varianta aktualizace je určena pro případ, kdy je spravováno více instalací systému. Navíc můžou různé instalace obsahovat různou skladbu rozšiřujících modulů. Pro tento účel slouží modul hromadné aktualizace, který přináší možnost vytvoření aktualizačních šablon. Tyto šablony sdružují instalace systému se stejnými rozšiřujícími moduly a na základě těchto šablon probíhá aktualizace. Díky tomuto přístupu je možné použit vytvořené CMS jako platformu pro vývoj a správu několika různých projektů.

Protože se jedná o modulární systém, nabízí se široké možnosti rozšíření. Jedním z možných rozšíření je implementace modulů pro správu menu a obrázků, jejichž požadavky jsou specifikované v této práci. Dalším zajímavým rozšířením by mohl být modul internetového obchodu. V modulu stránek by pak bylo například možné zpracovat návrh *layoutu* stránek pomocí *Drag and Drop* s použitím *javascriptové* knihovny jQuery.

Tato práce obsahuje kromě samotného popisu vývoje CMS i další užitečné informace. Čtenář se seznámí se základní charakteristikou aplikačního rámce Nette a databázové vrstvy Doctrine, je mu předložena implementace návrhového vzoru *singleton* a konceptu *hooků* a nakonec mu jsou popsány způsoby a prostředky pro ladění a testování webových aplikací.

Literatura

[1] Content Management System Distribution. *BuiltWith Technology Usage Statistics* [online]. [cit. 2011-12-31]. Dostupné z: http://trends.builtwith.com/cms

[2] System requirements. *Drupal* [online]. 2011 [cit. 2011-12-31]. Dostupné z: http://drupal.org/requirements

[3] Technical Requirements. SEVERDIA, Ron. *Joomla* [online]. 2011 [cit. 2011-12-31]. Dostupné z: http://www.joomla.org/technical-requirements.html

[4] Requirements. *WordPress* [online]. [cit. 2011-12-31]. Dostupné z: http://wordpress.org/about/requirements/

[5] POLZER, Jan. *Drupal*: *podrobný průvodce tvorbou a správou webů*. Vyd. 1. Brno: Computer Press, 2008, 262 s. ISBN 978-802-5119-464.

[6] RAHMEL, Dan. *Joomla!*: *podrobný průvodce tvorbou a správou webů*. Vyd. 1. Překlad Ondřej Gibl. Brno: Computer Press, 2010, 382 s. ISBN 978-802-5127-148.

[7] Framework/1.5. *Joomla Documentation* [online]. [cit. 2011-12-31]. Dostupné z: http://docs.joomla.org/Framework/1.5

[8] ZENDULKA, Jaroslav, Vladimír BÁRTÍK a Šárka KVĚTOŇOVÁ. *Analýza a návrh informačních systémů*: *Studijní opora*. Brno, 2006.

[9] Routování URL. *Nette* [online]. [cit. 2012-01-05]. Dostupné z: http://doc.nette.org/cs/routing

[10] Markup Validation Service. World Wide Web Consortium (W3C) [online]. [cit. 2012-01-05]. Dostupné z: http://validator.w3.org/

[11] Nette framework [online]. [cit. 2012-07-02]. Dostupné z: http://nette.org/cs/

[12] Passivate View [online]. [cit. 2012-07-26]. Dostupné z: http://www.martinfowler.com/eaaDev/PassiveScreen.html

[13] Model-View-* [online]. [cit. 2012-07-06]. Dostupné z: http://mnikoo.net/2010/06/14/model-view-star/

[14] Inversion of Control Containers and the Dependency Injection pattern [online]. [cit. 2012-07-10]. Dostupné z: http://martinfowler.com/articles/injection.html

[15] Seriál Doctrine 2 - Zdroják [online]. [cit. 2012-07-10]. Dostupné z: http://www.zdrojak.cz/serialy/doctrine-2/

[16] HosipLan/Nette-Doctrine-Sandbox [online]. [cit. 2012-07-15]. Dostupné z: https://github.com/HosipLan/Nette-Doctrine-Sandbox

[17] *OOP v PHP: Vzor Singleton* [online]. [cit. 2012-07-15]. Dostupné z: http://interval.cz/clanky/oop-v-php-vzor-singleton/

[18] Salted hash - další krok ke zvýšení bezpečnosti [online]. [cit. 2012-07-16]. Dostupné z: http://interval.cz/clanky/salted-hash-dalsi-krok-ke-zvyseni-bezpecnosti/

[19] Přihlašování oprávnění uživatelů: *Nette* [online]. [cit. 2012-07-17]. Dostupné z: http://doc.nette.org/cs/security

[20] Textile [online]. [cit. 2012-07-17]. Dostupné z: http://textile.thresholdstate.com/

[21] Easy MySQL Dump [online]. [cit. 2012-07-17]. Dostupné z: www.phpclasses.org/package/3868-PHP-Dump-the-structure-and-data-of-a-MySQL-database.html

[22] Phpsvnclient [online]. [cit. 2012-07-19]. Dostupné z: http://code.google.com/p/phpsvnclient/

[23] Json Rpc: Nette [online]. [cit. 2012-07-19]. Dostupné z: http://addons.nette.org/cs/json-rpc

[24] Json Debugování a zpracování chyb: *Nette* [online]. [cit. 2012-07-19]. Dostupné z: http://doc.nette.org/cs/debugging

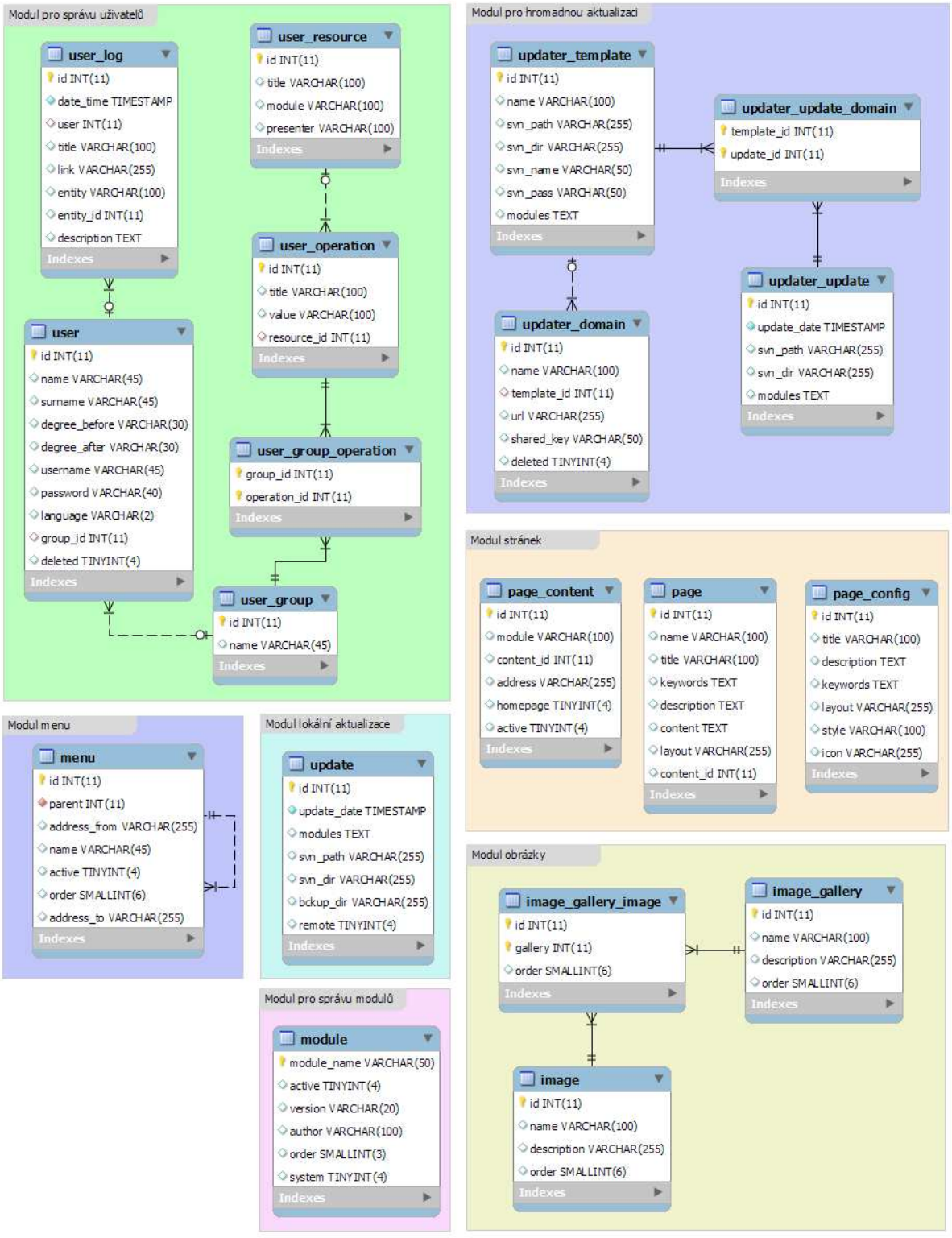
[25] Testování: *Nette* [online]. [cit. 2012-07-19]. Dostupné z: http://doc.nette.org/cs/testing

Seznam příloh

Příloha A: Obsah přiloženého CD

zprava.pdf – elektronická podoba tohoto dokumentu
manual.txt – textový dokument s návodem na instalaci CMS
cms – složka se zdrojovými soubory včetně všech potřebných knihoven

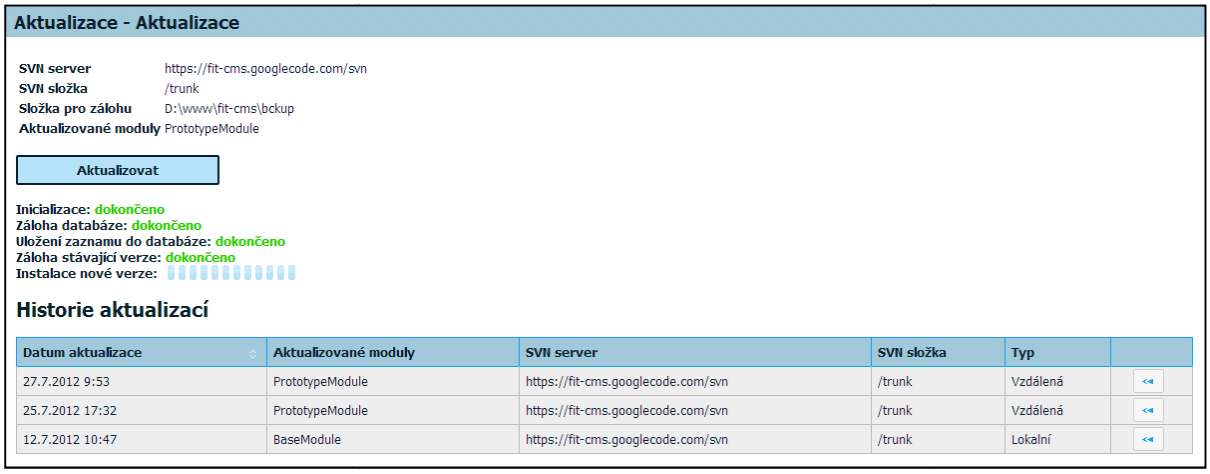
Příloha B: Fyzické schéma databáze CMS



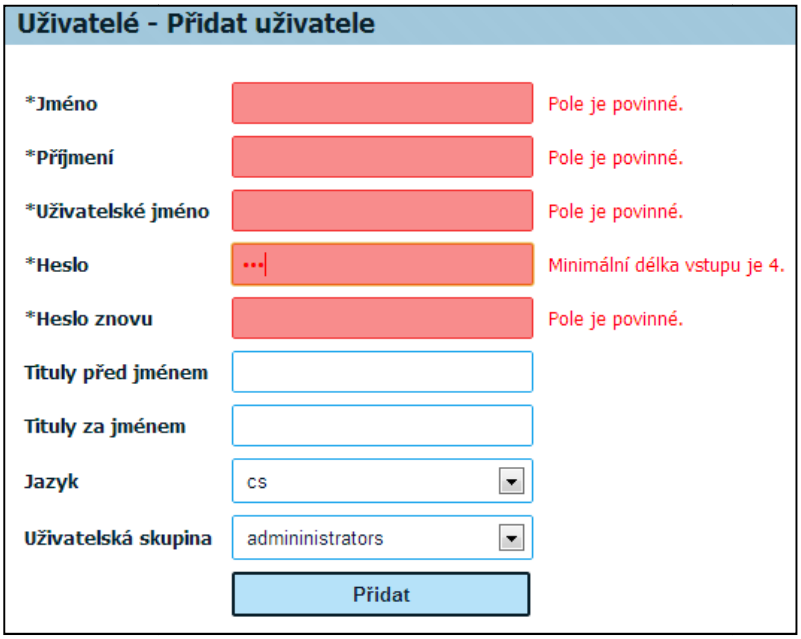
Obrázek 6: fyzické schéma databáze

Příloha C: Ukázky uživatelského rozhraní

Obrázek 7: uživatelské rozhraní

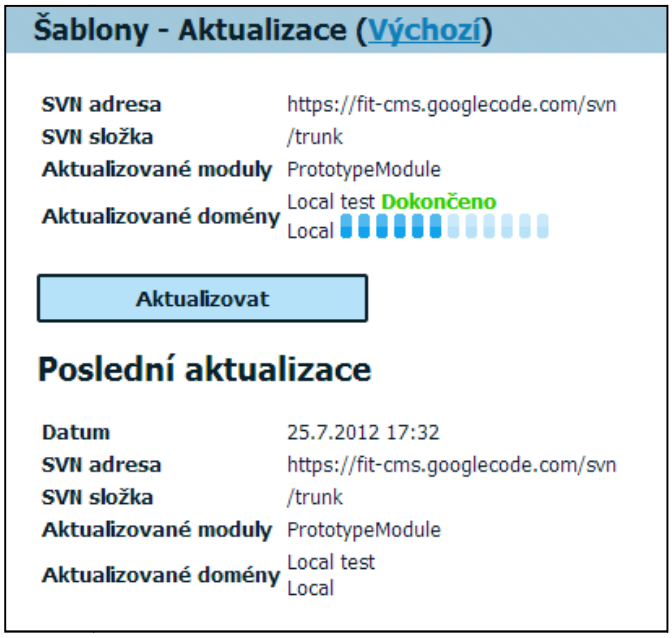


Obrázek 8: lokální aktualizace

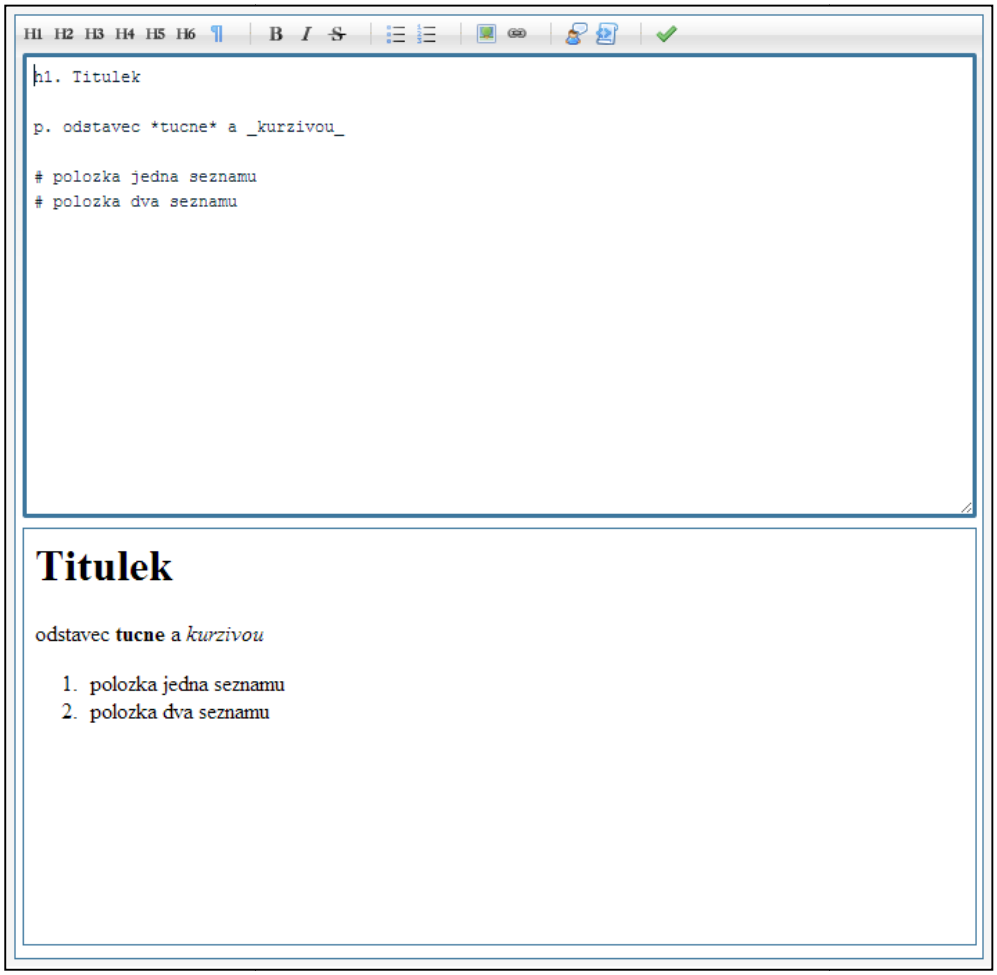


Obrázek 9: validace formuláře na straně klienta

Příloha D: Ukázky uživatelského rozhraní



Obrázek 10: hromadná aktualizace



Obrázek 11: WYSIWYG editor s náhledem