VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

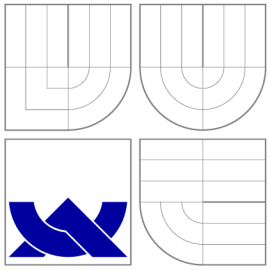
PLATFORMA PRO AUTOMATIZOVANÉ GENEROVÁNÍ INFORMAČNÍCH SYSTÉMŮ - PRODUKT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

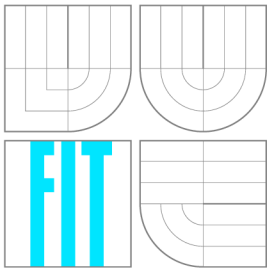
AUTOR PRÁCE
AUTHOR

LUKÁŠ GREŇ

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PLATFORMA PRO AUTOMATIZOVANÉ GENEROVÁNÍ INFORMAČNÍCH SYSTÉMŮ - PRODUKT

THE BASE FOR AUTOMATIC GENERATING THE INFORMATION SYSTEMS - PRODUCT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ GREŇ
AUTHOR

VEDOUCÍ PRÁCE — Ing. LUKÁŠ GRULICH
SUPERVISOR

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Greň Lukáš**
Obor: Informační technologie
Téma: **Platforma pro automatizované generování informačních systémů - Produkt**
Kategorie: Web

Pokyny:

1. Prostudujte existující implementace generických IS určených pro webové aplikace.
2. Navrhněte webový informační systém, který bude do detailu konfigurovatelný bez zásahu do zdrojových souborů. Konfigurací se rozumí zapnutí a vypnutí jednotlivých funkčních celků (modulů), nastavení vzhledu, pozice prvků na stránce a další. Systém by měl být konfigurovatelný tak, aby byl schopen vyhovět co nejširšímu spektru uživatelů s minimalizací nutnosti zásahu programátora. Systém bude maximálně automatizovaný, aby byl co nejméně náročný na lidské zdroje během provádění úprav pro zákazníka.
3. Zvolte vhodné programátorské prostředí a systém implementujte.
4. Systém testujte, zvláště pak kooperaci s generátorem.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Grulich Lukáš, Ing.**, UITS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Lukáš Greň**
Id studenta: 79023
Bytem: 17. listopadu 747/54, 708 00 Ostrava
Narozen: 03. 06. 1986, Ostrava-Vítkovice
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Platforma pro automatizované generování informačních systémů - Produkt
Vedoucí/školitel VŠKP: Grulich Lukáš, Ing.
Ústav: Ústav inteligentních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Zvyšující se poptávka po jednoduchých informačních systémech dává příležitost nabízet své systémy novým malým společnostem. Tyto vznikající společnosti si často vyvíjejí vlastní nízkorozpočtové produkty, které mnohdy kvůli nekvalitnímu návrhu a uspěchanému vývoji nelze jednoduše znovu použít. Cílem této práce je snížení nákladů při tvorbě a údržbě jednoduchých informačních systémů pomocí automatizace, modulárnosti a hromadné výroby se zachováním možnosti individuálních úprav.

Klíčová slova

informační systém, generování, generický, automatizace, modulárnost, Ruby, ActiveRecord

Abstract

The demand for simple information systems is increasing, giving opportunities to new small companies to sell their products. These emerging companies often develop their own low-cost systems, but mostly due to lack of proper design and hurried development is the reusability low and a new version is needed when new project begins. The goal of this thesis is reducing the cost of creating and maintaining simple information systems using automatization and modularity, allowing a mass production, but keeping a space for individual customizations.

Keywords

information system, generation, automatization, modularity, Ruby, ActiveRecord

Citace

Lukáš Greň: Platforma pro automatizované generování informačních systémů - Produkt, bakalářská práce, Brno, FIT VUT v Brně, 2008

Platforma pro automatizované generování informačních systémů - Produkt

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Lukáše Grulicha.

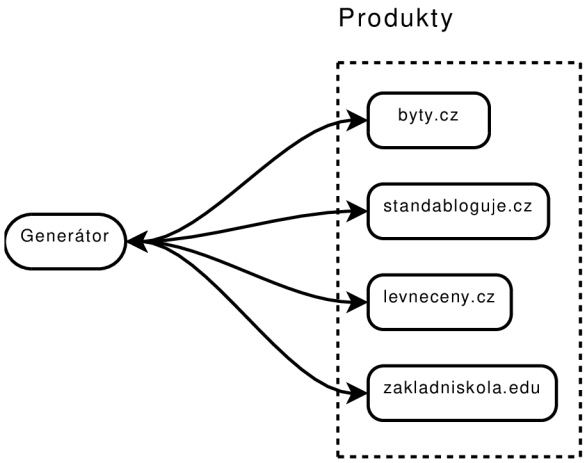
.....
Lukáš Greň
14. května 2008

Obsah

Kapitola 1

Úvod

Existuje mnoho zavedených malých firem zabývajících se vytvářením jednoduchých prezentací na internetu. Tyto firmy si většinou vytvářejí vlastní informační systémy a modifikují je podle potřeby. Vývoj kvalitního nového informačního systému je velmi nákladný, a protože do kvalitního systému malé firmy nechtějí investovat příliš prostředků, jsou systémy většinou nevalné kvality. Protože zisk z jedné malé zakázky je velmi nízký, další vývoj informačního sytému je pak v duchu zvyšování počtu funkcí na úkor kvality. Systém se pak dostává do stavu, kdy je dále neudržovatelný a jediný rozumný krok je vývoj nového čistého systému. Hlavním problémem takového směřování vývoje je neefektivní využití zdrojů a nedostatečné plánování.



Obrázek 1.1: Generátor vytváří a spravuje produkty

Tato bakalářská práce se snaží vytvořit nový obchodní model pro prodej informačních systému s co nejnižšími náklady na jednu zakázku. Základní myšlenka je vytvoření platformy pro automatizované generování informačních systémů, kde bude generátor vytvářet, upravovat a udržovat produkty - pro zákazníky sestavené informační systémy (obr. 1.1). Zákazník, který bude mít zájem o informační systém, zadá své požadavky v několika krocích a po odeslání je sestavený produkt ihned k dispozici. Takto si může výsledek svých požadavků okamžitě projít a popřípadě požadavky upravit. Pokud je uživatel spokojen, je mu oznámena výsledná měsíční částka, kterou bude za systém platit. Generátor pak bude schopen provo-

zovat stovky produktů a i s minimální marží by šlo o velmi výnosný podnik.

Náklady lze rozdělit na několik částí:

- Jednorázové náklady na vývoj
- Údržba základního systému
- Získání požadavků od zákazníka (schůzky, telefonáty)
- Náklady na úpravu systému pro konkrétního zákazníka
- Náklady na provoz a vznik jednoho systému (cena hw, energie, údržba)
- Marketing

Mnoho nákladů lze snížit určitou mírou automatizace. Cílem je vytvořit platformu pro automatizované generování informačních systémů, která bude sestavovat informační systémy dle specifikací získaných od zákazníka. Tato práce blíže rozvede možnosti snížení nákladu takovým směrem, aby byla zachována možnost individuálního přístupu ke každému produktu.

Kapitola 2

Stávající generické informační systémy

Na trhu existuje již několik generických systému s různým zaměřením. Mezi prvními generickými internetovými aplikacemi byly blogy. Poptávka po blogovacích systémech mnohonásobně převyšovala nabídku, a tak vznikalo mnoho firem zabývající se jejich tvorbou. Nejatraktivnější byly služby, které nabízely okamžité vytvoření blogu a co nejvíce přidaných služeb. V současnosti jsou tyto systémy velmi komplexní a tvoří ucelenou socialní platformu, mnohdy označovanou jako Web 2.0. Tyto platformy, které generují uživatelské systémy a řídí komunikaci mezi nimi, jsou velmi komplexní a jejich hodnota se pohybuje v řádech stovek miliónů až miliard dolarů [3]. Příkladem je komunitní portál MySpace.com, nebo český portál Lide.cz.

Méně provázané generické systémy jsou například Blogger nebo eStránky.cz. Obě platformy generují systémy, které jsou jednoduché, ale snadno specifikovatelné nezaškoleným uživatelem. Tyto a podobné systémy obsahují průvodce, který nezkušeného uživatele provede tvorbou svého blogu, stránky nebo jiné prezentace. Tyto systémy jsou většinou zdarma a zisk tvoří příjmy z reklamní činnosti, nebo je služba součástí strategie firmy, která se snaží pomocí neziskového projektu zvýšit návštěvnost svých jiných služeb. Systém, o kterém pojednává tato práce, by byl zaměřen primárně na paušálně platící zákazníky s možností každý systém maximálně upravit. Tato možnost je důležitá pro setrvání zákazníka u naší služby, pokud začne hledat komplexnější řešení. Systém, který bude stavět na principu této práce, tak musí nabídnout individuální přístup ke každému produktu.

Kapitola 3

Studie problému

Při návrhu platformy a produktu je nutné brát v úvahu nejen co nejnižší náklady na provoz a relativně nízké náklady na vývoj, ale také schopnost systému reflektovat individuální požadavky zákazníka, aniž by bylo nutné opustit politiku automatizovanosti.

Náklady lze rozdělit takto:

3.1 Jednorázové náklady na vývoj a údržbu sýstému

Kvalita platformy je stěžejní pro další úspěch projektu a proto zde snižovat náklady nelze. Naopak je rozumné maximalizovat investice do této části. Investice vynaložené zde i několikanásobně snižují celkové náklady.

Pokud rozdělíme vývoj na tři fáze: zpracování požadavků, architekturu a stavbu, maximální náklady na opravu problému mohou být až patnáctinásobné. Pokud se ale dostane chyba do závěrečného systémového testování a nebo se chyba objeví až po uvolnění produktu, mohou náklady na opravu narůst až na stonásobek částky, než kdyby byla chyba odhalena při svém vzniku (tabulka 3.1). Pokud je cílem snížení celkových nákladů, je důležité dbát na průběžné testování v jednotlivých fázích a dbát na výslednou kvalitu. Jak zajistit kvalitu při různých částech vývoje?

3.1.1 Zjištění vstupních požadavků

Abychom byli co nejlépe připraveni na různé požadavky zákazníků, je nutné vymezit segment trhu, na který se chceme zaměřit a shrnout charakteristiky požadavků na systémy, které jsou si pro většinu systémů podobné. Tyto charakteristky se budeme snažit implementovat, abychom byli schopni pokrýt většinu požadavků.

Tabulka 3.1: Průměrné náklady na opravu nedostatků [1]

Doba vzniku	Požadavky	Architektura	Stavba	Systémové testování	Nasazení
Požadavky	1	3	5-10	10	10-100
Architektura	-	1	10	15	25-100
Stavba	-	-	1	10	10-25

3.1.2 Architektura

Architektura musí být naprosto modulární a dobře zdokumentovaná tak, aby jakýkoliv další zásah neměl za následek snížení kvality systému. Dobrý návrh zvyšuje nejenom kvalitu softwaru, ale také znovupoužitelnost kódu.

3.1.3 Stavba

Při stavbě je výhodné použít pomocné nástroje, jako jsou například již hotové frameworky, knihovny a programovací prostředí, které ma výhodnou efektivitu příkazů.

3.1.4 Údržba systému

Pokud byly správně dodržovány předchozí kroky, náklady na údržby se radikálně snižují. Protože je ale vysoce pravděpodobné, že budou chyby nalezeny i po uvolnění produktu, kdy jsou náklady na opravu nejvyšší, je výhodné připravit postup, jak náklady na opravu takovýchto chyb co nejvíce minimalizovat. V případě této práce se jedná o automatické povyšování verzí v systému správy revizí. Tomuto problému se detailněji věnuje práce Ivety Šenfeldové [6].

3.2 Získání požadavků zákazníka a náklady na individuální řešení

Pokud chceme zjednodušit krok sběru požadavků zákazníka, musíme vytvořit systém, který jej krok po kroku provede co nejsrozumitelněji dotazy na specifikaci systému. Tento krok je velmi důležitý, protože chyby vzniklé v tomto bodě, odhalené až po uvedení systému do provozu, mohou zvýšit náklady na opravu až stokrát (tabulka 3.1). Pokud jsme při vývoji správně analyzovali obecné požadavky systémů v segmentu trhu, na který se chceme zaměřit, měly by být specifické požadavky jen úpravou již hotových komponent produktu.

3.3 Náklady na vznik a provoz jednoho systému

Při vzniku nového produktu je vhodná maximální automatičnost. Systém by se měl umět sám vygenerova a sestavit. Také instalace systému by měla být maximálně automatická. V optimálním případě dostane zákazník systém prakticky hotový ihned po zadání svých požadavků, pokud nebudou zahrnovat specifické úpravy a budou v reportoáru dosavadního systému.

Provoz serveru není tak finančně náročný jako lidské zdroje, a proto je přípustné zvýšit náklady v oblasti hardwaru, pokud se sníží jiné potřebné náklady. Jako příklad lze uvést využití interpretovaných jazyků, které sice zvýší náklady na provoz serverů, ale mnohonásobně sníží náklady na vývoj a údržbu softwaru.

3.4 Marketing

Marketing lze zcela vynechat, pokud bude služba kvalitní natolik, že ji budou zákazníci sami vyhledávat a pokud bude služba úspěšná v uživatelských recenzích. Podobně se rozšířilo povědomí o Google - *„Google se rozšířil doslova ústním podáním, jak jej spokojení uživatelé*

doporučovali svým přátelům, přičemž ti ostatní o něm vědeli z článků v médiích a na internetu. “[5]

3.5 Požadavky na produkt

Funkčnost bakalářské práce bude demonstrována na dvou systémech.

- Prezentace realitní kanceláře s jednoduchou správou nemovitostí
- Jednoduchý blog

Volbou těchto příkladů lze demonstrovat možnosti systému ve znovupoužitelnosti kódu pomocí modulárnosti a v možnosti nabízet systémy, které jsou cíleny pro různé skupiny zákazníků.

3.5.1 Blog

Uživatel jednoduchého blogu bude vyžadovat možnost jednoduchého psaní článků, jejich pozdější editaci či mazání. Správa obsahu by měla být přístupná v administraci a systém by měl umožňovat jednoduchou správu uživatelů. Pro komunikaci se čtenáři lze přikoupit například anketu.

3.5.2 Realitní kancelář

Zákazník, který by využíval informační systém pro nabídku nemovitostí, vyžaduje administraci se správou svých inzercí, jednoduchý redakční systém pro psaní novinek a jednoduchou správu uživatelů pro přístup do administrace.

3.5.3 Společné vlastnosti obou systémů

Oba systémy sdílejí mnoho společných vlastností. Uživatelé budou zajisté vyžadovat administraci, kde budou spravovat prezentovaný obsah. Správu uživatelů, která bude mít možnost registrovat a autentizovat uživatele. Jednoduchý redakční systém.

Kapitola 4

Příprava na stavbu aplikace a návrh

4.1 Programovací jazyk

Při výběru programovacího jazyka máme na vybranou mezi dvěma druhy: Kompilované jazyky a interpretované jazyky. Obě varianty mají své výhody i nevýhody:

4.1.1 Kompilované jazyky

Výhody:

- Vysoká rychlost
- Při použití typovaného jazyka se dostane méně chyb do produkce, protože mnoho problémů se odhalí již při překladu

Nevýhody:

- Nízká efektivita příkazů

4.1.2 Interpretované jazyky

Výhody:

- Konstrukce jsou v průměru mnohem jednodušší než v kompilovaných jazycích
- Mnoho hotových knihoven, časté použití v podobném případě ve světě, velká podpora
- Vysoká efektivita příkazů - kratší přehlednější kód

Nevýhody

- Nízká rychlost

V tuto chvíli jsou náklady na hardware nižší než náklady na lidské zdroje, a proto je výhodnější zvolit schůdnější jazyk pro programátora, než pro stroj. Proto jsme zvolili interpretované jazyky. Některé z možností jsou:

Tabulka 4.1: Efektivita příkazů ve vybraných jazycích [1]

Jazyk	Poměr k jazyku C
C	1
C++	2,5
Fortran 95	2
Java	2,5
Perl	6
Python	6
Smalltalk	6

PHP

Velmi oblíbený a nejvíce zastoupený scriptovací jazyk pro dávkové generování webových stránek. Nevýhodou je nízká flexibilita jazyka a nemožnost redefinic tříd za běhu programu. Často je PHP vybrán kvůli nízké ceně a snadné dostupnosti programátorů. To má ale často za následek velmi nízkou kvalitu výsledné implementace a tím i vysokou chybovost, která se mnohonásobně prodraží v pozdějších fázích projektu. Je mnohem výhodnější pracovat na projektu s kvalifikovanými zaměstnanci, kteří umí programovat „do“ jazyka, než „v“ programovacím jazyce [1]. Pak máme větší volnost a programátor nebude mít problém s jiným prostředím.

Python

Python je výkonný a hojně podporovaný programovací jazyk s nízkým zastoupením na trhu. Pokud bychom chtěli vytvořit aplikaci, která by neměla tolik funkcí, ale byla by směřována na vysokou návštěvnost a byla by založena na jednom produktu, byl by zvolen Python. Příkladem je firma Seznam.cz, která používá pro své systémy kombinaci C++ a Python umožňující vysokou rychlost a stále do jisté míry dobrou efektivitu příkazů.

Ruby

Jazyk Ruby klade důraz na co nejflexibilnější zápis konstrukcí a politiku „vše je objekt“. Jazyk podporuje maximální rozšiřitelnost, a to hlavně v případě redefinic, rozšířených definic, přejmenování názvu metod za běhu a přidávání nových funkcí do již definovaných tříd. Efektivita příkazů je velmi vysoká, jazyk je velmi pulární a tak hojně podporovaný komunitou.

ASP.NET

ASP.NET je jako jediný zde uvedený programovací jazyk proprietární a i když je ASP podporován i na webovém serveru Apache, řešení je experimentální a založit celý podnik na této neprověřené symbióze by byl risk. Proto by bylo nutné zakoupit i operační systém Windows Server, což by zvedlo náklady, ale přínos by byl minimální. Je zde také riziko skrytých nákladů, které si systém vyžádá v průběhu vývoje a provozu. S open source operačním systémem máme ze zkušeností jistotu nízkých nákladu a jednoduché správy, zvláště pak v případě Linux distribuce Ubuntu Server.

4.2 Databáze

Při výběru databáze bylo rozhodováno mezi dvěma kandidáty:

4.2.1 PostgreSQL

PostgreSQL nabízí více možností než MySQL například v podpoře kurzorů [2]. MySQL ale stále více PostgreSQL dohání a rozdíly již nejsou tak markantní [4]. Proti PostgreSQL mluví nižší popularita. V prvních fázích vývoje platformy pro automatizované generování informačních systémů byl vybrán PostgreSQL. V pozdějších fázích bylo využito komponenty ActiveRecord z frameworku Ruby on Rails, který umožňuje objektově relační mapování, kdy se rozdíly mezi jednotlivými databázemi v zásadě stírají. Pak je nejdůležitější kritérium podpora konektoru a celkový výkon řešení. Ten je podle několika zdrojů vyšší u MySQL.

4.2.2 MySQL

V prvních fázích vývoje byla databáze zavržena, a to hlavně z důvodu snahy o otestování i jiné databáze, než která je vyučována na VUT FIT. MySQL nenabízí tolik možností jako PostgreSQL, ale nepředpokládám, že by byly někdy využity. V moměntě, kdy bylo rozhodnuto o využití objektově relačního mapováni ActiveRecord z frameworku Ruby on Rails, byla možnost MySQL znovu zvážena z důvodů výskytu problémů při instalaci prostředí pro běh platformy na operačním systému GNU/Linux distribuce Ubuntu Server. Instalace by byla zajisté možná, ale vyžadovala by mnohem více času než instalace MySQL + ActiveRecord řešení.

4.3 Správa verzí

Platforma bude spravována v systému Subversion (dále jen SVN). Samotnému výběru systému pro správu revizí se věnuje práce Ivety Šenfeldové [6]. Využití správy revizí se v tomto případě pohybuje ve dvou rovinách.

- Podpora teamové spolupráce při vývoji
- Správa a generování produktů generátorem

4.3.1 Podpora teamové spolupráce při vývoji

Na této práci pracují dva studenti a obě práce spolu velmi úzce spolupracují. Pokud by nebyla nasazena správa revizí, bylo by mhoho prostředků promrháno jen pro režii výměny kódů a řešení vzniklých konfliktů a spojování práce na jednom souboru. Správa revizí umožňuje tuto režii snížit na minimum a velmi tak usnadňuje práci ve více lidech.

4.3.2 Správa a generování produktů generátorem

Způsobů generování produktů je více. Bylo by možné sestavovat stejný informační systém různě podle toho, z které URL přistupuje návštěvník. Pak by byl systém vždy aktuální a správa by byla jednodušší. Problém by ale nastal v případě, kdy by zákazník požadoval specifické úpravy. Ty bychom nemohli v rozumném rozsahu nabídnout, protože by se pak úpravy promítly do všech systémů, nebo by byla úprava skryta a využita jen pro konkrétního zákazníka, avšak robustnost systému by pak narůstala o jen zřídka používaný kód.

Druhou možností je vytvářet kopie základních systémů, které je pak možné libovolně upravovat. Tyto systémy lze pak aktualizovat na novější verzi právě s pomocí systému správy revizí Subversion. Vytvoření nové stránky tedy znamená vytvořit větev z nejnovější stabilní verze, tu upravit a následně exportovat do produkčního prostředí na webový server. V případě, že je vyvinuta nová stabilní verze, dříve vytvořené stránky jsou stále na sré verzi a je nutné manuálně vyvolat povýšení verzí. Při této operaci mohou nastat konflikty, kdy není možné automaticky verzi povýšit, protože úpravy pro specifického klienta zasahovaly do částí které jsou taktéž právě modifikovány ze stabilní verze. Zde je nutný zásah administrátora, kterému je ale celý proces ulehčen tím, že jdou předloženy jen postižené soubory přímo do HTML formuláře a administrátor tak konflikty vyřeší přímo na místě, kde probíhá povyšování. Celou problematikou správy revizí se mimojiné zabývá práce kolegyně Ivety Šenfeldové [6].

Kapitola 5

Architektura produktu

5.1 Postup při návrhu systému

Při návrhu systému bylo postupováno co nejpečlivěji tak, aby se případné problémy v návrhu odhalily co nejdříve a to z důvodu nákladů, které by musely být investovány v případě, kdy by se chyba dostala až do produkčního prostředí. Veškeré systémy byly vyvíjeny s ohledem na co nejnižší složitost, aby nedošlo k předimenzovanosti (overengineering). Kvalita byla proto upřednostňována nad kvantitou [1].

5.2 Modulárnost

Pro maximální škálovatelnost systému je nutné vytvořit systém modulů (komponent). Komponenta musí být odstranitelná z platformy tak, aby systém běžel bez chyb dál a měl za následek jen vypnutí závislých komponent.Komponenta musí mít možnost ovlivnit systém jen tím, že se vyskytuje v systému a není nutné ji nijak explicitně registrovat. Rozhodnutí, zda je systém danou komponentou ovlivněn, padá na ModuleManager, který má informace o tom, zda je běh komponenty povolen. Ne všechny součástí systému jsou komponenty. Součásti, které jsou neodstranitelné, a jsou pro běh systému vyžadovány vždy, jsou kategorizovány jako *vital*.

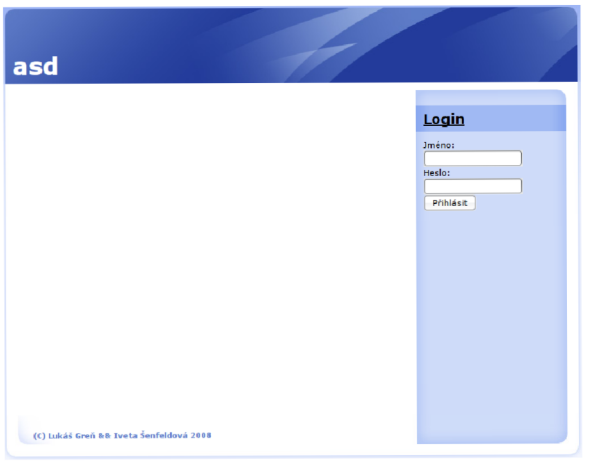
5.2.1 ModuleManager

ModuleManager je vital komponenta systému, která určuje, zda bude jiná komponenta (modul) inicializována, nebo ne. Komponenty během své inicializace nahlašují do systému, že v případě že nastane uržitá akce, chtějí zasáhnout. Pokud *ModuleManager* tuto inicializaci nepovolí, komponenty se neregistrují a akce proběhnou bez zásahu této komponenty. Komponenta však v systému stále existuje a její knihovna může být stále využívána. Komponenta se tak chová pasivně. *ModuleManager* zjišťuje, zda povolit inicializaci určité komponenty dotazem do vzdálené databáze generátoru, který shromažďuje informace o všech existujících produktech a jejich povolených komponentách. Pokud si například zákazník dokoupí nový modul, je změna zaznačena v centrální databázi generátoru. Komponenty jsou rozděleny na veřejné a soukromé. Veřejné komponenty jsou dostupné ke koupi zákazníkem, soukromé jsou vždy aktivní a pro zákazníka skryté. Výjimka je pak modul *ProductCustomizer*, který je součástí generátoru, a který pro svůj běh využívá samotný produkt jako framework. Tento modul je zaplý automaticky, pokud se produkt jmenuje „generator“.

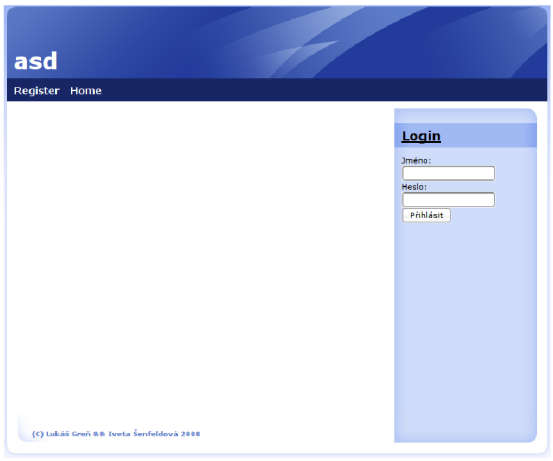
Každá nevitální komponenta obsahuje serializovaný objekt Module, který obsahuje vlastnosti modulu. Mimo jiné jeho závislosti, název, popis a zda je veřejný. Modul neobsahuje cenu, ta je řízena z centrální databáze produktu. Více o serializaci v kapitole 5.7.

Ukázka postupného zapínání modulů

Obrázek 5.1: Produkt bez veřejných modulů



Obrázek 5.2: Zapnut modul správy uživatelů



Obrázek 5.3: Zapnut modul menu

Obrázek 5.4: Zapnut modul administrace a redakční systém

Obrázek 5.5: Zapnut modul realitní kancelář a změna tématu

5.2.2 Loader - načítání knihoven

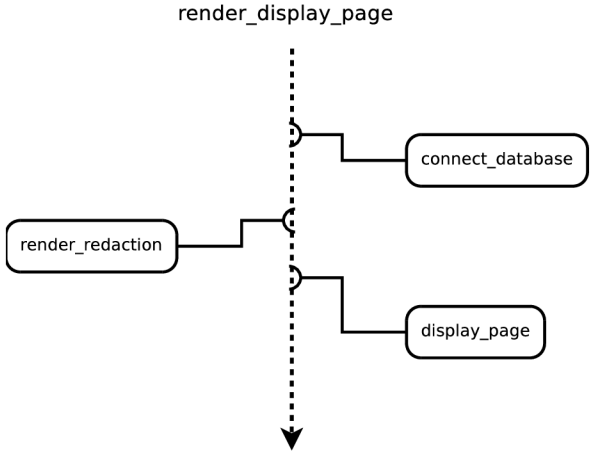
Loader je zvláštní vitální komponenta, která ma za úkol načíst všechny ostatní komponenty, ať už vitální nebo ne. Úkolem *Loaderu* je inicializovat jednotlivé knihovny a také se dotazovat *ModuleManagera*, zda má určitá public komponenta právo na spuštění inicializace. Obsahem loaderu jsou veškeré includy pomocných knihoven.

5.3 Systém háků a tyčí

Aby mohla komponenta ovlivnit chování systému, musí v produktu existovat z venku ovlivnitelný systém řízení reakcí na akce probíhající při běhu produktu. Pro akci v systému, která má být ovlivnitelná komponentou, byl zvolen pojem „tyč“ (anglicky „rod“). Pojem tyč byl vybrán proto, že se komponenty mohou registrovat pomocí svých „háků“ (anglicky hook). Registraci pak říkáme „hákování“. Tyto metafory podporují všeobecné porozumění problematiky [1].

V produktu existuje například tyč RenderPage (zobraz stránku), která reprezentuje akci zobrazení stránky. Pokud chce některá z komponent ovlivnit provádění této akce, vloží na *tyč* svůj *hák* na určitou pozici. V momentě, kdy je akce vyvolána, je tyč spuštěna a jsou vykonávány jednotlivé háky, které byly na tyč vloženy při inicializaci komponent. Pořadí *háků* na *tyčích* je důležité, a proto má komponenta při *hákování* (hooking) možnost definovat, mezi které háky bude umístěna. Pokud se například hákuje modul *Auth* (správa uživatelů) na hák *RenderPage*, uvádí, že musí být spuštěn až po připojení databáze a před samotným vykreslením stránky. Pokud jiný hák využívá správu uživatelů, definuje, že má být spuštěn před hákem Auth. Pomocí těchto omezení je pak vytvořeno pořadí háků, které vyhovuje podmínkám. Může existovat více vyhovujících variant.

Modul má také schopnost odhákovat kterýkoliv hák na jakékoliv tyči. Například v případě modulu *ThemeSwitcher*, jenž má za úkol měnit grafické ztvárnění systému, modul vyhákuje již stávající hák *load_default_theme* a nahradí jej svým *theme_switch*.



Obrázek 5.6: Příklad hákování

Ukázka inicializace modulu, kdy se modul hákuje na tyč:

```
Rod.get("render_display_page").unhook("load_default_theme");

Rod.get("render_display_page").hook(
    Hook.new("switch_theme") do
        ThemeSwitcher.instance.hook_theme_switch
    end, "render_display_page")

Rod.get("render_display_page").hook(
    Hook.new("themeswitcher_render") do
        ThemeSwitcher.instance.hook_render
    end,["render_display_page"], ["switch_theme"])
```

V prvním příkazu si modul vyzvedne *render_display_page*. Pak na tomto háku zavolá metodu *unhook* a odhákuje hák *load_default_theme*. Háky a tyče mají své identifikace podle jména. Tyče jsou spravovány statickými metodami třídy Rod.

Druhý příkaz znova vyzvedne tyč *render_display_page* a zavolá její metodu hook. Parametr metody hook je objekt Hook a pozice, již tvoří dva další parametry, a to výčet jmen háků před a za aktuálně vkládaným hákem. Háky ve výčtech nemusí být ve chvíli hákování přítomné, podle těchto pravidel se budou hákovat i nově příchozí háky tak, aby platily i dříve uvedená pravidla.

Konstruktor objektu *Hook* požaduje dva parametry - název háku a tělo funkce, která se má vykonat po spuštění háku. V tomto těle může být jakákoliv posloupnost příkazů, ale je doporučeno uvést jen volání funkce knihovny modulu s prefixem *hook_*.

5.4 Systém zobrazování informací nezávisle na konkrétní grafické interpretaci

Pro umožnění diversifikace grafického ztvárnění zobrazených informací jsou do jisté míry oddělené zobrazovací a logické prvky.

5.4.1 RenderBlock

Komponenty generují bloky informací, *RenderBlocky*, které jsou shromažďovány v modulu *Render* a v momentě, kdy se má vykreslit stráka, jsou veškeré nashromážděné bloky odeslány do modulu *Theme*. Modul Theme je komponenta, která příchozí *RenderBlocky* umístí na stránku. Produkt může obsahovat více modulů Theme, každý pak může mít vlastní metodiku prezentování RenderBloků. Modul Theme, kam bude *Render* přeposílat RenderBlocky, je vybírán modulem *ThemeSwitcher*.

Například modul *Redaction* vytvoří *RenderBlock* s obsahem článků, který má být umístěn na titulní straně. Tento vytvořený blok ihned po vytvoření odešle do Renderu. V momentě, kdy se začne provádět vykreslování stránky, je *RenderBlock* se články odeslán do některého z modulů Theme. Theme příjme RenderBlocky a následně začne zpracovávat předpis struktury stránky. Podle struktury stránky pak umístí články na logicky správné místo na stránce.

Každý *RenderBlock* je definovaný svým jménem, třídami, prioritou a scriptem. Jméno jednoznačně identifikuje blok a je pak možné manipulovat s každým blokem zvlášť. Blok můžeme umístit do tříd a tím definovat skupiny, do kterých blok patří. Lze tak manipulovat s více bloky najednou v rámci jedné šablony. Blok může patřit do více tříd, chování je obdobné CSS třídám class. Script bloku určuje jeho obsah a priorita pořadí bloků ve třídě (nižší číslo má vyšší prioritu). Je doporučeno obsah formátovat pomocí XHTML a používat CSS.

Způsob, jakým komponenta vytvoří script RenderBlocku je ponechán na ní, ale doporučený způsob je využití ERB šablonovacího systému. Šablonovací systém odděluje logickou a prezentační vrstvu a velmi zpřehledňuje kód. Modul vždy připraví data do private proměnných své třídy a připojí šablonu takzvaným bindingem. Binding zpřístupní proměnné z daného objektu do celé šablony. Kód v logické vrstvě může vypadat takto:

```
@articles = RedactionArticles.find(:all)
template = ERB.new('templates/administration_articles.erb'))
block = RenderBlock.new("redaction_administration_articles")
block.script = template.result(Redaction.instance.get_binding)
block.classes = ["content"]
Render.instance.render_block_add(block)
```

Kód šablony pak vypadá například takto:

```
Seznam článků:
<%@articles.each { |article|%>
    <div>
        <%=article.text%>
    </div>
<%}>
```

Úkolem logické vrstvy je získat data (@articles = ...), načíst šablonu (template = Erb.new....) a zpřístupnit data (Redaction.instance.get_binding). Pak je vše připraveno na zpracování šablony a uložení výstupu do *RenderBlocku*, respektive jeho scriptu. V šabloně lze používat programovací jazyk Ruby, pokud je kód uveden ve značkách `<\% \%>`.

5.4.2 Předpis struktury stránky(layoutu) v Theme

Aktuální stránka systému je definována pomocí dvou atributů - *Page* a *PageSection*. *PageSection* udává layout, který má být použit. Pokud se například nacházíme v administraci, je PageSection nastaven na *Administration*, a vybraný layout je proto administration. Layout obsahuje XHTML a speciální makra, které definují *Containery*. *Container* je prostor pro zobrazení *RenderBlocků*. Pomocí příkazu:

```
DISPLAY_BLOCK nazevrenderblocku
```

udáváme, že má být na místě příkazu zobrazen block s názvem *nazevrenderblocku*. Pokud chceme zobrazit v *Containeru* více bloků spadajících do určité třídy, použijeme příkaz:

```
DISPLAY_BLOCKS trida1 trida2 trida3
```

kdy definujeme seznam tříd, které chceme zobrazit. V tuto chvíli přichází do hry priority jednotlivých RenderBlocků, které určí výsledné pořadí zobrazení.

Výsledný kousek layuoutu pak může vypadat takto:

```
<div id="page-in">
    <div id="left">
        <div id="left-in">
            #DISPLAY_BLOCK menu#
        </div>
    </div>
    <div id="right">
        <div id="right-in">
            #DISPLAY_BLOCKS panel#
        </div>
    </div>
 </div>
```

RenderBlock je zobrazen vždy maximálně jednou a to ve speciálnějším případě. Spadá-li tedy modul *menu* do třídy panel, ale je definován příkaz #DISPLAY_BLOCK menu#, je blok umístěn do tohoto prostoru a v Conteineru #DISPLAY_BLOCKS panel# uveden nebude.

5.5 ActiveRecord

V průběhu návrhu bylo rozhodnuto o využití objektově relačního mapování ActiveRecord, na kterém staví populární a kvalitní framework Ruby on Rails. Pomocí objektově relačního mapování je možné odstínit specifika jednotlivých databází a zapouzdřit do obecného systému objektů. Je pak možné poměrně jednoduše přecházet na jiné databáze v případě potřeby.
Výhody objektového zapouzdření:

- Jednoduché spojení databáze s použitým programovacím jazykem
- Žádné míchání syntaxí různých jazyků (Ruby + SQL)
- Objekty napojené na databázi lze jednoduše doplnit dalšími funkcemi
- Zkrácení psaného kódu

Nevýhody objektového zapouzdření:

- Chybí plná kontrola nad databází
- Dotazy nelze plně optimalizovat (ActiveRecord do jisté míry sám optimalizuje)
- Některé dotazy nelze vyjádřit objektovým přístupem

Ne všechny dotazy lze vyjádřit pomocí dotazů na objekty, zde však ActiveRecord nabízí možnost objektově relační mapování opustit a doplnit pomocí přímo vkládaného SQL.

ActiveRecord je také vhodný pro samotnou tvorbu databáze, kterou využívají migrace (kapitola 5.6). Lze tak definovat strukturu databáze nezávisle na dodavateli databázového řešení.

5.6 Migrace databáze

Jak jsem již zmiňoval, abychom podchytili aplikaci změn v systému na jednotlivé produkty, využívá správu revizí SVN. Ta si ale není schopná poradit s vývojem databáze a aplikací jejich změn na nižší verze tak, aby došlo k povýšení na novější. Tento problém musí být ošetřen explicitně v rámci produktu a musí existovat způsob, jak verzi databáze aktualizovat a synchronizovat se zdrojovým kódem, který je již verzován pomocí SVN. Proto byl vytvořen systém migrací, inspirovaný migracemi z Ruby frameworku Ruby on Rails. Framework Ruby on Rails umožňuje programátorovi vytvořit pro každou verzi migraci, ve které definuje úpravy databáze pro zvýšeni verze směrem o jednu nahoru. Definuje také změny po sestup verze směrem dolů. Pro případ této práce je migrace z Ruby on Rail nevhodná, protože je jeden celek a v našem případě je definice databáze roztříštěn do modulů. Roztříštěnost definice struktury databáze umožňuje maximální modulárnost systému. V tomto případě je proto systém migrací upraven tak že:

- Bude definována jen cesta povyšování a degradace bude vypuštěna
- Každý modul bude mít svuj migrační script

Pokud bude mít každý modul svůj migrační script, je nutné nějak zajistit pořadí, ve kterém budou spouštěny. K tomu lze s velkou výhodou použít stávající systém háků a tyčí (kapitola 5.6). Jednotlivé migrace se při zavádění modulů nahákují na migrační tyče. Každá verze databáze, na kterou se bude migrovat dostává svoji tyč. Tyče jsou pak spouštěny v vzestupném pořadí od aktuální verze databáze do cílové. Pokud tedy například v systému existují migrace [1, 2, 3, 4, 9], aktuálně je databáze na verzi 3 a chceme migrovat na databázi 9, jsou spuštěny tyče 4 a 9.

5.7 Serializace objektů

Pro maximální zjednodušení úkládání persistentních dat mimo databázi je v co největší míře používána serializace objektů. Objekt je uložen v nativním Ruby formátu YAML do souboru a v případě potřeby znovu načten. Serializace se používá zejména v ukládání nastavení produktu a pro uložení objektu do Session. Serializace probíhá jednoduše příkazem

```
object.to_yaml
```

a deserializace

```
object = YAML::load(file)
```

Další použití serializace je v uložení informace o modulu. Protože každá revize v databázi má svůj set modulů a protože se pracuje s více revizemi zároveň, není praktické ukládání informací o modulech do jedné databáze, ale je výhodné vždy načíst aktuální vlastnosti modulu z SVN. Informace o modulu jsou serializovány v module.yaml v každém adresáři komponenty a jsou vytaženy příkazem

```
specifikace = YAML::load(‘svn cat svn://server/cesta/k/module.yaml‘)
```

Ze specifikace modulu je možné zjistit jeho závislosti na jiných modulech, popis a název.

5.8 Realizace konkrétních modulů

Výběr konkrétních modulů k implementaci ovlivnil výběr systémů k demonstraci produktu - blogu a realitní kanceláře.Oba systémy budou využívat pro formátování výstupu informaci značkovací jazyk XHTML - tím se zabývají komponenty Render (kapitola 5.8.1) a Theme (kapitola 5.8.2). Komunikaci s uživatelem zapouzdřuje modul Input, který spravuje vstupy GET, POST a SESSION. Systém uživatelských vstupů byl upraven do podoby jazyka PHP.Systém zobrazování informací je detailně popsán v kapitole Systém zobrazování informací nezávisle na konkrétní grafické interpretaci (kapitola 5.4). Informace jsou uloženy většinou v databázi. Připojením a obsluhou databáze se zabývá modul Database (kapitola 5.8.4). Schéma databáze je většinou triviální a nemá smysl jej uvádět. Záměrem této práce není vytvoření konkrétního systému, ale proces jeho stavby, obecná podpůrná vrstva produktu(framework) a demostrace funkčnosti pomocí jednoduchých modulů.

5.8.1 Zobrazení - Render

Hlavním úkolem modulu Render je podpora zobrazování výstupních informací ze systému. Render udává aktuální stránku, na které se systém nachází. Ostatní moduly se informují o této stránce a podle toho generují RenderBlocky (kapitola 5.4.1). Informace o stránce je rozdělená na dva parametry - Page a PageSection. PageSection udává, ve které sekci se systém nachází (například frontend, administration), page udává stránku v rámci sekce. Render v průběhu své inicializace vytváří tyč *render_display_page*, na kterou se hákují ostatní moduly, které chtějí provést svou akci v průběhu vykreslování stránky. V okamžiku kdy je systém inicializován a je možné začít se zobrazováním, je tyč *render_display_page* spuštěna a jsou provedeny veškeré nahákované události.

```
Rod.get('render_display_page').run
```

Tyč je spuštěna a dostávají se do hry moduly, které byly nahákovány na tuto tyč v průbehu inicializace. Tyto moduly si například zjistí aktuální Page a PageSection a podle toho vytvoří RenderBlock, ten následně odešlou do Render a ten, v momentě kdy příjde na řadu, bloky přepošle do vybraného Theme. Postup volání akcí na tyči *render_display_page* může vypadat takto:

```
Spoustim rod ‘render_display_page‘
    render_display_page - Hook run:  connect_database
    render_display_page - Hook run:  authorize
    render_display_page - Hook run:  inquiry_render
    render_display_page - Hook run:  product_customizer
    render_display_page - Hook run:  auth_render
    render_display_page - Hook run:  administration
    render_display_page - Hook run:  prototypejs_header
    render_display_page - Hook run:  redaction
    render_display_page - Hook run:  menu
    render_display_page - Hook run:  switch_theme
    render_display_page - Hook run:  themeswitcher_render
    render_display_page - Hook run:  render_display_page
```

V prvních fázích dojde k připojení databáze a autorizace uživatele (kapitola 5.8.3). Následně je zpracována anketa, která generuje RenderBlock do Renderu. Mezi dalšími je například

také zpracován modul redakce a menu. Jako poslední akce je zavolán hák render_display_page, který všechny nashromážděné moduly odešle do Theme.

5.8.2 Grafické témata - Theme

Modul Theme je zodpovědný za interpretaci RenderBlocků a jejich umístění na stránku. Theme umisťuje bloky na stránku podle svého layoutu (více o layoutech v kapitole 5.4 Systém zobrazování informací nezávisle na konkrétní grafické interpretaci. Toto chování není vynucené a lze jej předefinovat. V systému může existovat více grafických témat a tvůrce nového tématu může využít implicitní chování, nebo vytvořit kompletně nové chování modulu Theme. Pro změnu tématu, kam přeposílá Render RenderBlocky slouží modul ThemeSwitcher, který je nepovinnou součástí systému. Pokud není ThemeSwitcher zapnut, je použit vždy implicitní Theme.

5.8.3 Správa uživatelů - Auth

Modul Auth slouží k autentifikaci a autorizaci uživatele. Identifikuje uživatele na základě přihlašovacího jména a hesla, které přichází skrz modul Input. Pokud je autentifikace úspěšná, je jméno a heslo uloženo do Input proměnné SESSION, která jako jediná zachovává svůj obsah i po novém požadavku na server. Session lze využít všude tam, kde není žádoucí nestavovost protokolu HTTP. Pokud jsou data nutná pro autentifikaci uložena do Session, není nutné od uživatele požadovat identifikaci pokaždé, kdy se dotáže na informace ze serveru, ale jen jednou za určitý čas. Pokud neproběhne autentifikace úspěšně, je nastavena stránka Renderu na Page = not_authorized a PageSection = frontend. Pokud je tedy autentifikace ověřována na tyří Render_display_page před ostatními moduly a autorizace neproběhne v pořádku, následnující moduly dostávají stránku not_authorized místo skutečného požadavku návštěvníka

Modul Auth plně využívá ActiveRecord. Definuje třídu User, která dědí z ActiveRecord::Base. Pokud následně voláme metody podle specifikace ActiveRecord nad touto třídou a nad instancemi této třídy, dotazujeme se přímo na data v databázi. Základní ActiveRecord třída je odekorována o další metody, například *visitor*. Metoda *visitor* funguje podobně jako singleton. Vrací vždy jeden objekt reprezentující aktuálního návštěvníka. Objekt reprezentují aktuálního návštěvníka pak tedy získáme příkazem.

```
navstevnik = User.visitor
```

5.8.4 Databáze - Database

Při zavádění modulu databáze je komponenta nahákována na tyč *render_display_page* jako hák s názvem *connect_database*. Pokud se hák spustí, je prvně zjištěno, zda existuje databáze, na kterou se chceme připojit. Pokud databáze neexistuje, je vytvořena nová databáze spolu s tabulkou *meta*, která obsahuje záznam o verzi celé databáze. Ve chvíli vytváření databáze je verze nastavena na 0. Dále je spuštěna migrace (kapitola 5.6).

5.8.5 Systémový záznam - Log

Třída pro zaznamenávání dění v systému, podpůrný nástroj pro odlaďování chyb. Soubor s logem je ukládán do adresáře *var*, který je určen pro přechodné uchovávání souborů, které nejsou spravovány systémem revizí a nemají vliv na chod systému.

5.8.6 Uživatelský vstup - Input

Abychom skryli implementační charakteristiky modulu CGI, byla vytvořena třída Input, která zpouzdřuje uživatelský vstup. Modul CGI pro Ruby není dle mého názoru velmi programátorsky přívětivý, a to z toho důvodu, že není možné vyzvednout data ze vstupu GET, pokud přicházejí data POST. Toto bylo ve třídě Input podchyceno a data z GET jsou rozparzovány z URL explicitně mimo rozhraní CGI. Takto je vytvořeno prostředí podobné například PHP nebo ASP.

5.8.7 Navigace - Menu

Komponenta Menu je určena k tvorbě navigace na stránce. Navigace samotná nemá definované své prvky, ale implementuje metodu add_item, která umožňuje ostatním modulům přidání nového odkazu do navigace. Modul posléze odesílá svůj vygenerovaný RenderBlock do Renderu, kde promítá předchozí přidané položky. Přidání nové položky musí být provedeno před odesláním RenderBlocku, a proto na to musí být myšleno při zavádění háků. Například modul Render vkládá své položky do navigace, a proto hák, ve kterém k tomuto přidávání dochází musí proběhnout dříve, než hák samotného zpracování menu.

```
Spoustim rod ‘render_display_page‘
    render_display_page - Hook run:    connect_database
    render_display_page - Hook run:    administration
    render_display_page - Hook run:    redaction
    render_display_page - Hook run:    menu
    render_display_page - Hook run:    render_display_page
```

Zde vidíme, že hák *menu* je až za hákem *redaction* a *administration*, kde oba *menu* využívají. Takové nahákování, které má podmínku, že musí být umístěno před menu vypadá v případe modulu *Redaction* takto:

```
Rod.get('render_display_page').hook(
    Hook.new('redaction') do
        Redaction.instance.hook_render
    end,['render_display_page', 'menu'], ['connect_database', 'auth'])
```

Poslední parametr udává množinu háků, před kterými musí být aktuálně hákovaný hák uveden. Druhá množina udává háky, před kterými musí být hák umístěn.

5.8.8 Administrace - Administration

Vzhledem k tomu, že administrace nemusí být součástí všech stránek zákazníků, byla administrace navržena a implementována jako modul. Modul Administrace využívá Menu, kam vkládá položku pro vstup do administace. Administrace samotná není složitý modul - jejím úkolem je zajistit přepnutí PageSection na administration. Příslušné moduly následně reagují tvorbou jiných RenderBlocků.

5.8.9 Redakční systém - Redaction

Dle odhadů velmi často používaným modulem bude redakční systém Redaction. Ten je v odevzdávané verzi velmi jednoduchý. Lze přidávat, upravovat a mazat články a je evidován autor článku.

5.8.10 Anketa - Inquiry

Tento modul je použitelný zejména tam, kde se uplatňuje redakční systém. Je vhodný pro oba naše praktické příklady, a proto byl implementován.

5.8.11 Realitní kancelář - Estate

Estate je modul realitní kanceláře, který umožňuje zákazníkům publikovat nabídku nemovitostí a jejich ceny. Jde jen o velice jednoduché řešení demonstrující, že je opravdu možné na produktu založit více různých informačních systémů s různým zaměřením.

Kapitola 6

Závěr

Vzhledem k rozsáhlosti problematiky automatizovaného generování bylo mnoho problémů vyřešeno pouze okrajově, ale přesto rozsah implementace nabízí možnost posouzení, zda je přístup ke snižování nákladů pomocí automatizace určitých činností správný. S určitostí můžeme konstatovat výhody systému pouze po uvedení do reálného provozu, přesto zastávám názor, že přínosy by převažovaly nad zápory a systém by byl profitabilní.

6.1 Další směrování vývoje

Pokud by byl systém komerčně úspěšný, vývoj by dále pokračoval v těchto oblastech:

6.1.1 RenderBlock

RenderBlocky obsahují XHTML a tím také omezují výstupní zobrazovaný jazyk na XHTML. Pokud by bylo usouzeno, že mohou vzniknout požadavky na jiný výstup než HTML, bylo by nutné navhrnout a implementovat vlastní obecný značkovací jazyk, který by byl použit k dopravě informace z jednotlivých modulů do Theme. Theme samotné může veškeré informace interpretovat jakýmkoliv způsobem a může kompletně opustit politiku layoutů. Jediný problém tedy zůstává obsah XHTML v RenderBlocích.

6.1.2 Input

Třída Input implementuje vstup ze strany uživatele jen s využitím rozhraní CGI. Pokud by bylo po systému požadován provoz na jiných technologiích a s jinými uživatelskými vstupy, musel by být Input přepracován a to nejlépe do podoby, kdy by byl vstup POST a GET spojen do jednoho obecného uživatelského vstupu. Byly by však velmi zasaženy stávající implementace.

6.1.3 Výkon systému

Pokud by byl zaznamenán enormní zájem o produkty, bylo by nutné vyřešit vzrůstající požadavky na výkon hardwaru. Systém by bylo možné provozovat na více serverech s minimální úpravou stávajících zdrojových kódů. Musel by však být implementován přerozdělovač zátěže, který by nové produkty umisťoval na méně výkonné servery. Druhá možnost jak zajistit zvýšení výkonu a vyhnout se úpravám stávajícího systému je využití virtualizace. Ta vytvoří z více strojů jeden virtuální a zátěž je plynule přerozdělována na nízké úrovni. Nevýhodou je však režie takového systému.

6.1.4 Správa stránek

Správa stránek je roztříštěna po celém systému a není tak možnost jednoduše sestavit mapu stránek a jednotlivé stránky spravovat. Bylo by tak vhodné navrhnout centrální systém stránek, do kterého by například pomocí hákování jednotlivé moduly registrovaly své stránky. Každá stránka by pak měla svou tyč, která by byla spuštěna spolu s render_display_page.

Literatura

[1] Steve McConnell. *Dokonalý kód.* Computer Press, 2006. ISBN 80-251-0849-X.

[2] Bruce Momjian. *PostgreSQL praktický průvodce.* Computer Press, 2003. ISBN 80-7226-954-2.

[3] WWW stránky. News corporation. `http://www.newscorp.com/news/news_251.html`.

[4] WWW stránky. Wikivs. `http://www.wikivs.com/wiki/MySQL_vs_PostgreSQL`.

[5] David A. Vise and Mark Malseed. *Google Story.* Pragma, 2005. ISBN 978-80-7349-034-8.

[6] Iveta Šenfeldová. *Platforma pro automatizované generování informačních systémů - Generátor.* Brno, FIT VUT v Brně, 2008.