VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

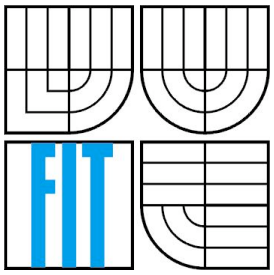
REZERVAČNÍ A UBYTOVACÍ SYSTÉM V RUBY ON RAILS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE JOSEF HOVAD
AUTHOR

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

REZERVAČNÍ A UBYTOVACÍ SYSTÉM V RUBY ON RAILS

RESERVATION AND ACCOMMODATION SYSTEM IN RUBY ON RAILS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — JOSEF HOVAD
AUTHOR

VEDOUCÍ PRÁCE — Ing. DUŠAN VRÁŽEL
SUPERVISOR

BRNO 2008

Zadání bakalářské práce/6630/2007/xhovad01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Hovad Josef**
Obor: Informační technologie
Téma: **Rezervační a ubytovací systém v Ruby on Rails**
Kategorie: Web

Pokyny:

1. Seznamte se s programovacím jazykem Ruby a s moderními přístupy objektového návrhu pro webové aplikace (návrhové vzory, rámce). Důkladněji se seznamte s rámcem Ruby on Rails.
2. Seznamte se s požadavky kladenými na Rezervační a ubytovací systémy a prostudujte již existující řešení.
3. Navrhněte vícejazyčnou webovou aplikaci pro katalogizaci, administraci a rezervaci ubytovacích zařízení. Při návrhu využijte modelovacích prostředků UML. Rozsah aplikace konzultujte s vedoucím.
4. Aplikaci implementujte v Ruby on Rails a ověřte její funkčnost na vhodně zvoleném vzorku dat.
5. Zhodnoťte dosažené výsledky, zejména výhody a nevýhody použití frameworku Ruby on Rails a porovnejte vaše řešení s již existujícími ubytovacími systémy.

Literatura:

- Informace a dokumentace dostupné na http://www.**ruby**on**rails**.org/.
- Thomas D., Hansson H.: Agile Web Development with Rails: Second Edition. Pragmatic Bookshelf 2006.

Při obhajobě semestrální části projektu je požadováno:

- Body 1. až 3.

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: **Vrážel Dušan, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

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

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Josef Hovad**
Id studenta: 78987
Bytem: U Lipek 571, 561 64 Jablonné nad Orlicí
Narozen: 21. 02. 1986, Ústí nad Orlicí
(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: Rezervační a ubytovací systém v Ruby on Rails
Vedoucí/školitel VŠKP: Vrážel Dušan, Ing.
Ústav: Ústav informačních 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

Práce je zaměřena na webové informační systémy. Především na použití inženýrských postupů při jejich návrhu a na použití moderních technologií při jejich vývoji, což je demonstrováno na systému zajišťujícím rezervaci ubytování implementovaném ve frameworku Ruby on Rails.

Klíčová slova

informační systém, web, rezervace, ubytování, návrhový vzor, framework, ruby, ruby on rails

Abstract

The thesis is aimed at web information systems. Primarily, it is aimed at the usage of an engineering process of their design and for the purpose of development of modern technologies. This is illustrated on the reservation system of accommodation, which is implemented in Ruby on Rails framework.

Keywords

information system, web, reservation, accommodation, design pattern, framework, ruby, ruby on rails

Citace

Hovad Josef: Rezervační a ubytovací systém v Ruby on Rails. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Rezervační a ubytovací systém v Ruby on Rails

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Dušana Vrážela. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Josef Hovad
10. 5. 2008

Poděkování

Děkuji vedoucímu diplomové práce Ing. Dušanu Vráželovi, za velmi užitečnou metodickou pomoc a cenné rady při zpracování bakalářské práce. Za podporu a pochopení děkuji rodině, zvláště mamince a přátelům.

Obsah

Úvod

Tato práce je zaměřena na webové informační systémy. Především na použití inženýrských postupů při jejich návrhu a na použití moderních technologií při jejich vývoji, což je demonstrováno na systému zajišťujícím rezervaci ubytování.

První část práce je věnována charakteristice řešené problematiky. Je zde objasněna problematika návrhových vzorů pro návrh webových aplikací a charakteristika frameworků, zvláště pak frameworků pro vývoj webových aplikací. Následuje charakteristika jazyka Ruby a popis frameworku Ruby on Rails, ve kterém je vyvíjen modelový systém.

Druhá část je věnována představení obdobných existujících systémů a především návrhu modelového webového rezervačního a ubytovacího systému. Při návrhu je užito modelovacích prostředků UML.

Ve třetí části jsou popsány významné části samotné implementace modelového systému.

Čtvrtá část práce je věnována zhodnocení dosažených výsledků v kontextu použití frameworku Ruby on Rails a porovnání vytvořeného systému s již existujícími rezervačními a ubytovacími systémy.

1 Charakteristika řešené problematiky

1.1 Formulace cílů práce

Cíle bakalářské práce Rezervační a ubytovací systém v Ruby on Rails jsou:

1. Seznámení s programovacím jazykem Ruby a s moderními přístupy objektového návrhu pro webové aplikace jako jsou návrhové vzory a frameworky neboli rámce. Především potom důkladné seznámení s frameworkem Ruby on Rails.
2. Seznámení s požadavky kladenými na rezervační a ubytovací systémy s prostudováním již existujících řešení.
3. Navrhnutí vícejazyčné webové aplikace pro katalogizaci, administraci a rezervaci ubytovacích zařízení s použitím modelovacích prostředků UML.
4. Implementace navržené aplikace ve frameworku Ruby on Rails a ověření její funkčnosti na zvoleném vzorku dat.
5. Zhodnocení dosažených výsledku, zejména výhody a nevýhody použití frameworku Ruby on Rails a porovnání výsledného řešení s již existujícími rezervačními a ubytovacími systémy.

1.2 Moderní přístupy objektového návrhu webových aplikací

Tato kapitola pojednává o základech řešené problematiky. Konkrétně obsahuje seznámení s problematikou návrhových vzorů a frameworků v kontextu webových aplikací.

1.2.1 Návrhové vzory

Informační systémy, coby jeden z největších fenoménů poslední doby, zaznamenávají stále větší rozvoj a díky tomu se dostávají do stavu, kdy často naráží na hranice únosné složitosti své architektury. To je závažný problém, protože na návrhu správné architektury téměř vždy závisí úspěšné dokončení a provoz celého systému. Přitom se však lze při návrhu často setkat s opakujícími se problémy, neboť informační systémy jsou si často velmi podobné a na výjimky narazíme zřídka. Na základě toho vznikají návrhové vzory, které zavádí do návrhu architektury systému standardní postupy zvyšující přehlednost a srozumitelnost. Nejedná se tedy o knihovny či jiné části zdrojových kódů (algoritmy), ale o metodiky a šablony pro řešení určitého typu problému, který se však může objevit v kontextu různých situací. Například objektově orientované návrhové vzory řeší vztahy mezi třídami nebo objekty, nespecifikují však finální podobu daných tříd a objektů. Více viz [4].

1.2.2 Návrhové vzory pro návrh webových aplikací

Budeme-li se dívat na web z pohledu informačního systému, nalezneme stejně jako u informačních systémů obecně, množství opakujících se problémů, které je třeba při návrhu webových informačních systémů řešit.

Na webovou aplikaci se pak můžeme dívat jako na několik vzájemně spolupracujících komponent, například zpracovávání dat z formulářů, navigace, operace s databází, autentifikace, správa chybových hlášení, e-commerce.

Nebo jako na vrstvy, tzn. prezentační vrstva, vrstva řízení chodu aplikace, vrstva manipulací s daty.

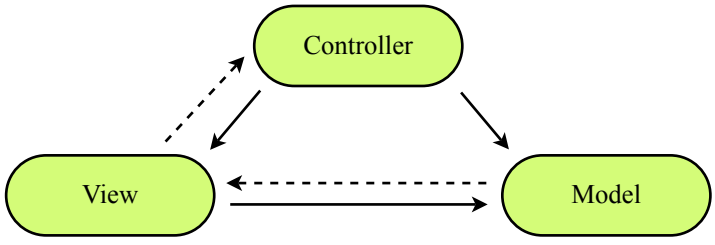
Nejen tyto problémy jsou společné pro většinu webových aplikací a lze na ně aplikovat vhodné návrhové vzory, které standardizují a zpřehledňují řešení. Zde jsou uvedeny dva obvykle využívané koncepty:

1.2.2.1 „Model 1“

Jde o jednoduchý koncept, oddělující aplikační logiku a zobrazení obsahu. V jedné části aplikace tak probíhají veškeré manipulace s daty a řízení chodu aplikace, druhá část se stará čistě o zobrazení dat, které příjme od první částí. V případě webu tento přístup typicky oprostí kód o části spojené s používáním značkovacího jazyka, čímž dojde ke značnému zpřehlednění. Přesto je tento koncept doporučován pouze pro implementaci projektů menšího rozsahu.

1.2.2.2 „Model 2“, Model-View-Controller

Nejčastěji používaným návrhovým vzorem webových aplikací je Model-view-controller (model–pohled–řadič) zkráceně MVC. Vychází z předchozího konceptu „Model 1“, ovšem rozděluje řízení chodu aplikace a manipulaci s daty na dvě samostatné části – controller (řadič) a model, viz obr. 1. Vzor MVC popsal Trygve Reenskaug již v roce 1979. Použit byl v jazyce Smalltalk a touto implementací bylo inspirováno mnoho dalších projektů.



Obr. 1

Model je část aplikace, která řeší veškeré manipulace s daty a zapouzdřuje tak věcnou a datovou logiku systému. Často se jedná o nejsložitější část systému, ke které je vhodné přistupovat přes ucelené aplikační rozhraní (API). Typicky se jedná o ukládání dat zadaných uživatelem, jejich editaci a v neposlední řadě transformaci.

View, překládaný jako *pohled*, je část aplikace, která se stará o samotné zobrazení grafického rozhraní pro interakci s uživatelem. V případě webu je grafické rozhraní typicky realizováno značkovacím jazykem z rodiny SGML (HTML). V této části je tedy oddělena prezentační logika od řídící a datové. To umožňuje udržovatelnost prezentační vrstvy, kde jsou obvyklé četné požadavky na změny.

Controller, překládaný jako *řadič*, je část aplikace, která reaguje na události (požadavky uživatele) v prezentační vrstvě, a na jejich základě požaduje služby od části Model. Jedná se o vrstvu, která leží mezi prezentační (view) a aplikační (model) a tím redukuje jejich vzájemnou závislost.

Obvykle pracuje návrhový vzor MVC následujícím způsobem:

1. Dojde k události vyvolané uživatelem (např. odeslání formuláře).
2. Je odeslána informace části controller.
3. Controller zašle požadavek na model a na jeho případnou aktualizaci (např. přidání položky do katalogu).
4. Model zpracuje změněná data (např. uloží novou položku do katalogu a přepočítá celkový počet položek v katalogu).
5. View použije aktualizovaný model pro zobrazení aktualizovaných dat na výstup.
6. Čeká se na další událost, která opakuje popsaný cyklus.

1.2.3 Frameworky

Cílem frameworku, překládáno jako *rámec*, je podpora a usnadnění vývoje a organizace softwarových projektů. Může obsahovat knihovny připravených funkcí, API, podpůrné programy, návrhové vzory či doporučené postupy pro vývoj. Přebírá tak rutinní problémy v dané oblasti a tím dovoluje vývojáři soustředit se pouze na samotnou problematiku vyvíjeného produktu.

Dalo by se říci, že na rozdíl od běžné knihovny, která vývojáři poskytuje opakovaně použitelnou funkcionalitu, framework poskytuje především opakovaně použitelné rozhraní. Při práci s knihovnou jsou volány funkce, které knihovna poskytuje, naopak framework vývojář nikde volat nemusí, princip je opačný.

Framework můžeme obvykle rozdělit na tzv. hot spots a frozen spots. Frozen spots definují celkovou architekturu vytvářeného systému, která je neměnná v každém nasazení frameworku. Hot spots jsou pak části frameworku, které programátor používá pro vytváření konkrétní aplikace a mají tím pádem vždy jinou podobu. Pro hot spots je většinou vyhrazen konkrétní prostor.

Efektivita použití frameworku se projevuje především při jeho opětovném nasazení, kdy již je vývojář s frameworkem seznámen. Vývoj pak může probíhat až několikrát rychleji.

1.2.4 Frameworky pro webové aplikace

Jedná se o frameworky vytvořené přímo pro vývoj dynamických webových aplikací. Většina z nich vychází z návrhového vzoru MVC. Základními rysy frameworku pro webové aplikace, které usnadňují vývoj, jsou např. následující.

- Přístup k relační databázi a její mapování. Frameworky obvykle poskytují jednotné API pro přístup k systému řízení báze dat, čímž mezi systémem řízení báze dat a aplikací vytváří vrstvu a aplikace není závislá na použití konkrétního typu systému řízení báze dat. Objektově orientované frameworky typicky automaticky mapují záznamy z relační databáze na objekty a opačně.
- Mapování URL. Automatickým přepisováním parametrů do přívětivých URL se systém stává použitelnější a navíc dosahuje lepších výsledků ve vyhledávačích.
- Šablonování. Každá stránka webu obvykle obsahuje neměnné části (záhlaví, zápatí) a části, které se dynamicky mění (hlavní obsah). Proto se používají šablony. Do částí stránky, které jsou dynamické, lze vypisovat obsahy proměnných nesoucí data získaná z databáze, ostatní části jsou neměnné. Aplikace díky šablonám získá konzistentní vzhled.
- Existují frameworky, které mají zabudovaný autentizační a autorizační mechanizmus, který umožňuje zabezpečený přístup uživatelů k funkcím aplikace na základě daných pravidel.
- Cachování je technika, která ukládá dokumenty (či jejich části) do mezipaměti webového serveru, čímž podstatně redukuje zátěž serveru. Některé frameworky poskytují mechanizmy pro kontrolu cachování a případné přemosťování procedur spojených s prezentací dat přes šablony.
- AJAX je technika pro vytváření interaktivních webových aplikací. Některé frameworky poskytují zabudované JavaScript frameworky pro snadné vytváření aplikací s použitím technologie AJAX.

V souvislosti s frameworky pro vývoj webových aplikací jsou často zmiňovány také Content Management Systems, zkráceně CMS, překládané jako *systémy pro správu obsahu*. Jejich typickým představitelem je CMS Drupal, Joomla nebo Plone. Jejich základním cílem je „správa obsahu webové aplikace“, ale poskytují množství dalších možností. Je možné rozšiřovat jejich funkcionalitu moduly, které lze vytvářet prostřednictvím propracovaných API a knihoven funkcí. Při vývoji je vyžadováno dodržování standardů. Tím CMS naplňují přibližně stejnou skupinu požadavků, které jsou obvykle kladeny na frameworky. Konkrétně CMS Drupal lze považovat za systém pro správu obsahu skoro tak jako za framework pro vývoj webových aplikací.

1.3 Seznámení s programovacím jazykem Ruby

Ruby je interpretovaný skriptovací programovací jazyk. Jedná se o plně objektově orientovaný jazyk výjimečný především jednoduchou syntaxí. Tvůrcem jazyka je jediný člověk – Yukihiro Matsumoto, který při práci na Ruby sloučil vlastnosti svých oblíbených jazyků, kterými jsou Perl, Smalltalk, Eiffel, Ada a Lisp.

Od zveřejnění v roce 1995 si Ruby získává stále větší popularitu. TIOBE index, podle kterého se měří popularita programovacích jazyků, řadí Ruby celosvětově na devátou pozici. O velký rozmach se zasloužil především framework Ruby on Rails a také to, že Ruby je zcela svobodný open source produkt, který může kdokoli zdarma používat, kopírovat, modifikovat a distribuovat.

Pro nalezení ideální syntaxe se Yukihiro Matsumoto inspiroval z ostatních jazyků. Chtěl skriptovací jazyk, který bude robustnější než Perl a více objektově orientovaný než Python. V Ruby je

vše objektem. Zcela každý prvek může mít své instanční proměnné a metody. Tato síla je často demonstrována kouskem kódu, kde se volá metoda čísla:

```
5.times { print "Hallo world!" }
```

Ve většině ostatních jazyků nejsou čísla a ostatní základní datové typy objekty. Ruby tak vychází z jazyka Smalltalk přiřazováním metod a instančních proměnných všem datovým typům.

Ruby umožňuje vývojáři volně upravovat určité své části, které mohou být odstraněny nebo předefinovány. Například operátory jsou metodami (byť s výjimečnou syntaxí), čímž mohou být snadno předefinovány. Více viz [5].

1.3.1 Odlišnosti jazyka Ruby

1.3.1.1 Bloky

Mezi jazyky, z kterých je Ruby inspirováno, je mimo jiné Lisp. Ten byl inspirací ke vzniku bloků v Ruby. V Ruby není blok jen určitá ohraničená část kódu. Blok je část kódu, který je předáván metodě jako parametr. Bloky jsou tedy zdrojem vysoké flexibility. Zde je malá ukázka:

```
cities = ["Praha", "Brno", "Ostrava" ]
cities.each do |element|
  puts element
end
```

V této ukázce je blokem část kódu ohraničená klíčovými slovy do ... end. Ten je předáván jako parametr metodě *each*, která prochází jednotlivé prvky pole *cities* a tiskne je na standardní výstup. Podobně nechává programátorovi prostor pro upřesnění detailů mnoho metod v Ruby, stejně jako může definovat vlastní metody pracující s bloky. Pak je v definici metody použito klíčové slovo *yield* v místě, kde se má kód bloku provést. Více viz [2].

1.3.1.2 Moduly a Mixiny

Na rozdíl od mnoha jiných objektově orientovaných jazyků Ruby poskytuje pouze jednoduchou dědičnost. Nicméně k dispozici je koncept *modulů*, na které se lze dívat jako na kolekce metod. Do třídy tak může být začleněn modul a mohou být volně využívány všechny jeho metody. Vývojáři Ruby je tento koncept považován za čistější řešení než vícenásobná dědičnost, která je v tomto případě řešena např. začleněním více modulů do třídy. Metody přejaté ze začleněného modulu se pak stávají instančními metodami třídy. Tato technika je nazývána *mixing in* a modul je označován jako *mixin*. Více viz [2].

1.3.2 Oblast možného použití jazyka

Oblast použití Ruby je široká, stejně jako u jiných skriptovacích jazyků. Na systémech typu UNIX tak může usnadňovat každodenní práci a díky podpoře regulérních výrazů například práci s textem. Nicméně díky přehledné syntaxi se stejně dobře hodí i pro rozsáhlé projekty, CGI skripty nebo dokonce GUI aplikace (wxRuby atd.).

1.4 Ruby on Rails

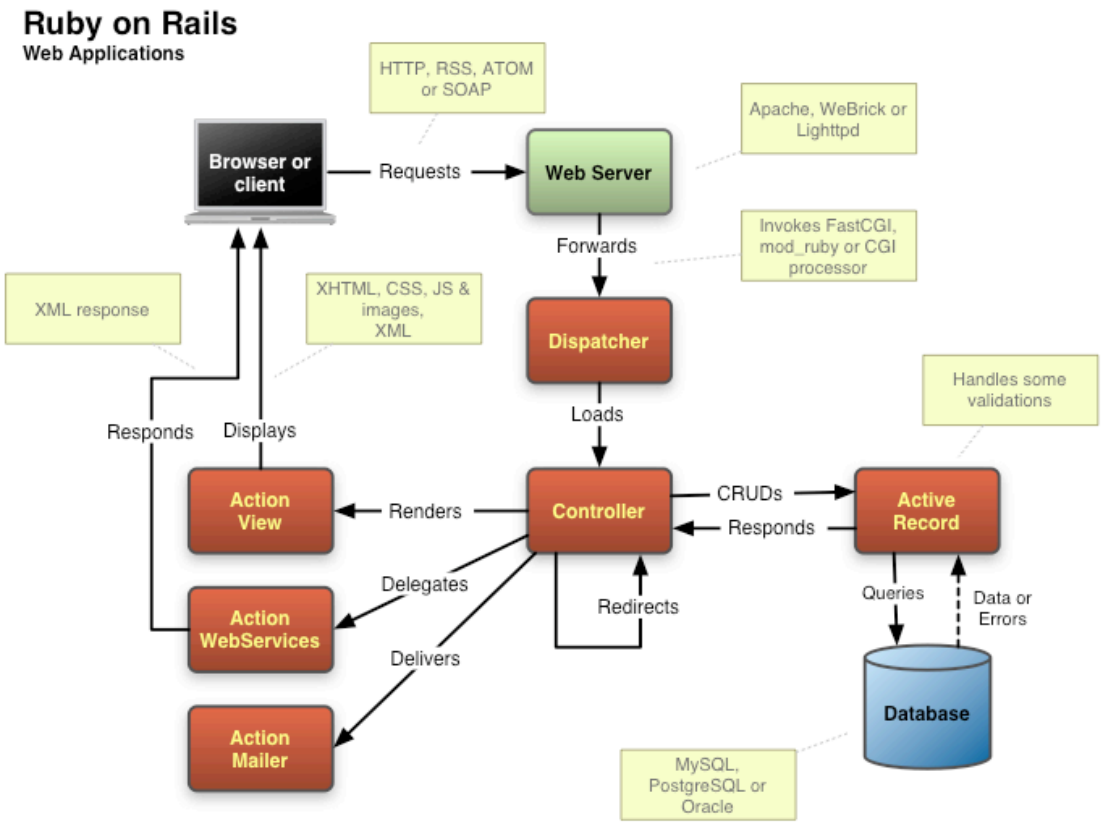
Ruby on Rails je framework pro vývoj webových aplikací vytvořený v jazyce Ruby. Jeho autorem je David Heinemeier Hansson. Ruby on Rails jsou postaveny podle návrhového vzoru Model-View-Controller.

Hlavní principy, ze kterých Ruby on Rails vychází, jsou dva. Především **upřednostňování konvence před konfigurací**. Programátor tedy konfiguruje pouze části aplikace, které vybočují z obvykle užívaných vzorů. A princip „Don't Repeat Yourself", tedy „neopakovat se", což znamená, že každá informace je v aplikaci uložena pouze jednou. Např. v třídě pro práci s tabulkou relační databáze tedy není nutné definovat použité „sloupce", framework si tuto informaci zjistí sám.

Ruby on Rails je distribuováno s nástrojem *scaffold*, s jehož pomocí lze snadno vytvořit základní podobu komponent model, controller a view. Součástí distribuce je dále také webový server *WEBrick* a *Rake*, což je obdoba známého *make*. Součástí frameworku je JavaScript framework Prototype a knihovna Script.aculo.us pro snadnou práci s technologií AJAX a vizuálními efekty.

Framework Ruby on Rails je rozdělen do balíčků nazvaných Active Record, Active Resource, Action Pack, Active Support, Action Mailer a Action Web Services. Dále je možné Ruby on Rails rozšiřovat prostřednictvím pluginů.

Z diagramu, viz obr. 2, je patrná infrastruktura frameworku, diagram zachycuje tok každého typu požadavku a odpovědi na něj:



Obr. 2

1.4.1 Active Record

Názvem Active Record je označován návrhový vzor implementující techniku tzv. objektově-relačního mapování. V Ruby on Rails je Active Record třídou implementující *Model* podle tohoto návrhového vzoru a tvoří tak základ každé aplikace v Ruby on Rails. Potažmo se jedná o hlavní benefit tohoto frameworku.

Active Record mapuje tabulky relační databáze na třídy a jednotlivé řádky tabulek na objekty, tedy instance těchto tříd. Sloupce tabulek jsou mapovány na atributy objektů. V Ruby on Rails navíc Active Record poskytuje ošetření vztahů mezi tabulkami relační databáze (many-to-one, many-to-many apod.) a validaci dat zadávaných uživatelem. Tím vývojáři odpadá nekonečné psaní stejných částí kódu jako `SELECT * FROM products WHERE id = 1`. V Ruby on Rails napíše jednoduše `Product.find(1)`. Již na této zcela jednoduché ukázce je patrné zpříjemnění práce vývojáře. Následující kód demonstruje změnu ceny u produktu:

```
product = Product.find(1)
product.price = 32
product.save
```

V porovnání s řešením stejného úkolu přímým zápisem SQL dotazů je benefit Active Record markantní. Díky zmiňovanému ošetřování vazeb mezi tabulkami lze podobným způsobem načítat data, která získáváme s přímým SQL dotazem s klauzulí JOIN, viz následující ukázka.

```
orders = Order.find(:all,
  :conditions => ["name like ?", params[:name]+"%"],
  :order => "pay_type, shipped_at DESC",
  :limit => 10
  :include => "products")
```

1.4.2 Action Pack

Ve vzoru MVC jsou *view* a *controller* úzce spolupracující komponenty. Proto jsou v Ruby on Rails implementovány v jednom balíku zvaném Action Pack. Neznamená to však, že by byl *view* a *controller* spojen v jeden celek, jak by se mohlo zdát. Z pohledu vývojáře jde o zcela oddělené části, přesně podle návrhového vzoru MVC. Jejich spojení je patrné pouze na úrovni implementace frameworku.

View, v souladu se vzorem MVC, realizuje veškeré zobrazování obsahu v prohlížeči. Může jít pouze o část HTML kódu, většinou je však skrze pohled vypisován dynamický obsah, který je podobně jako v jiných webových frameworcích generován skrze šablony. Ruby on Rails poskytuje šablony pro tři typy výstupů.

Standardní šablony, označené příponou *rhtml*, poskytují možnost kombinovat kód HTML a Ruby, pomocí kterého se vkládají dynamické části výstupu. Tento přístup je běžný ve spoustě webových systémů. Šablony označené koncovkou *rxml* slouží pro generování XML výstupu. Do třetice šablony s koncovkou *rjs*, které usnadňují práci s JavaScriptem a technologií AJAX.

Controller opět plně respektuje filozofii MVC. Navíc poskytuje několik užitečných služeb. Mezi ně patří zajištění převodu URL na vnitřní řídící prvky aplikace, správa cachování a správa seassons.

Více viz [1] a [3].

2 Návrh systému

2.1 Požadavky kladené na rezervační a ubytovací systémy

Reservační a ubytovací systémy tvoří kategorii systémů užívaných v oblasti cestovního ruchu. Nabízejí katalogizované přehledy objektů, v nichž je nabízeno ubytování a jejich prostřednictvím lze ubytování elektronicky rezervovat.

Typickými uživateli rezervačních a ubytovacích systémů jsou majitelé objektů, ve kterých je nabízeno ubytování (např. hotely nebo penziony). Dále pak agentury, které zprostředkovávají ubytování v těchto objektech a samozřejmě samotní hosté, kteří hledají vhodné ubytování a provádějí rezervace.

Potenciální hosté využívající rezervační a ubytovací systém mají většinou stejné požadavky. Potřebují především snadno získat dostatek informací o objektu, nejlépe s doprovodnými fotografiemi. Pokud se jedná o systém nabízející více objektů, požadují patřičnou katalogizaci. Je také podstatný snadný průchod procesem rezervace objektu.

Rozličné požadavky na rezervační a ubytovací systém mají samotní hoteliéři a zprostředkovatelské agentury, které případný systém provozují a poskytují ubytování ve více nezávislých objektech.

Pokud má systém sloužit právě pro jeden hotel či penzion, jeho majitel očekává možnost komplexní správy všech informací o svém podniku. Jsou to tedy došlé rezervace, jejich historie, napojení na účetní systém, nebo zapouzdření funkcí zajišťujících provoz účetnictví přímo do ubytovacího a rezervačního systému. Dále by měl systém nabízet možnosti organizovat práce v objektu. Tzn. pokud se jedná o větší hotel, může to být například zápis zaměstnanců na směny, organizace činností jako úklid pokojů apod.

Požadavky agentury, která zprostředkovává ubytování ve více nezávislých objektech, se pak zaměřují především na chod její vlastní činnosti a nezahrnují potřebu organizovat objekty interně. Mají ovšem zvýšené požadavky na možnost katalogizace jednotlivých objektů a ucelenou administraci i pro případy, kdy se prostřednictvím systému zprostředkovává ubytování v objektech různých typů (např. v hotelech, penzionech, chatách, kempech).

2.1.1 Existující řešení

Z existujících řešení byli vybráni následující zástupci, prezentující hlavní formy podob systému, které lze zařadit do kategorie rezervačních a ubytovacích systémů.

Hotelový rezervační a recepční systém *Previo* - previo.cz. Previo je webová aplikace, určená provozovatelům menších a středních ubytovacích zařízení hotelového typu (hotel, penzion, apartmány, kemp apod.). Poskytuje možnosti pro kompletní správu interního běhu ubytovacího zařízení.

Rezervační systém pro přeprodejce, přímé prodejce a veřejné klienty PEAR s.r.o - codoma.cz. Poskytuje seznam a vyhledávání volných kapacit ubytovacích zařízení, kalkulace cen, správu klientů a možnost rezervace. Systém je vhodný pro agentury a cestovní kanceláře.

Rezervace ubytování Limba - limba.com. Rezervační systém agentury zprostředkovávající ubytování především v chatách a chalupách. Poskytuje rozsáhlý strukturovaný katalog objektů s možností rezervace.

2.2 Návrh řešení

Následující kapitola je návrhem vícejazyčné webové aplikace pro katalogizaci, administraci a rezervaci ubytovacích zařízení s použitím modelovacích prostředků UML.

2.2.1 Určení systému

Vytvářený systém bude určen pro agenturu, která zprostředkovává ubytování v objektech různých typů. Mezi hlavní požadavky patří katalogizace objektů, správa objednávek, možnost přímé editace objektů jejich majiteli apod.

2.2.2 Slovník pojmů

Pro práci na projektu rezervačního a ubytovacího systému byla zvolena následující terminologie.

Objekt

Nemovitost nabízená k pronájmu. Může jít o hotel, penzion, chatu, kemp apod. Objekt může být dále členěn na menší jednotky, podle toho o jaký typ objektu se jedná.

Návštěvník

Jedná se o potenciálního zákazníka ubytovacího zařízení a potažmo zákazníka agentury provozující rezervační a ubytovací systém, který **není do systému přihlášen**.

Registrovaný uživatel

Jedná se o potenciálního zákazníka ubytovacího zařízení, který má v rezervačním a ubytovacím systému svůj uživatelský účet a je do systému přihlášen.

Majitel objektu

Majitelem objektu je majitel objektu nebo jeho zástupce, který je oprávněn spravovat prezentaci objektu v systému.

Sekretariát

Je osoba z agentury provozující rezervační a ubytovací systém, která spravuje systém na základní úrovni.

Administrátor

Kompletně spravuje systém.

2.2.3 Funkční požadavky

Na základě konzultace s vedoucím bakalářské práce a s osobou pohybující se v oblasti ubytování, byly stanoveny následující funkční požadavky na vznikající systém.

Návštěvník systému může:

- prohledávat katalog;
- filtrovat a řadit výsledky;
- zobrazit detaily o objektu jako popis objektu, základní cenu, ceny v sezónách, fotografie, majitele objektu, hodnocení objektu;
- procházet strukturou objektů a zobrazovat detaily případných částí objektu (pokojů);
- zaregistrovat se do systému a stát se *registrovaným uživatelem*, nebo se přihlásit pod existujícím uživatelským účtem.
- zvolit lokalizaci

Registrovaný uživatel může provádět operace příslušící *návštěvníkovi systému* a navíc:

- hodnotit objekt;
- přidat objekt do skupiny svých oblíbených objektů;
- objednat objekt;
- zobrazovat a editovat detaily svého účtu;
- na stránce s detaily svého účtu zobrazit seznam svých zadaných objednávek a jejich stav, tzn. zda je objednávka potvrzena či zamítnuta.

Majitel objektu může provádět operace příslušící *návštěvníkovi systému* a navíc:

- zobrazit seznam vlastněných objektů a zcela je editovat;
- přidávat nové objekty a členit je na další menší jednotky;
- prohlížet objednávky vztahující se k jeho objektům;
- potvrzovat nebo zamítat objednávky vztahující se k jeho objektům;
- zobrazit seznam sezón zahrnující probíhající sezónu a nastávající sezóny;
- přidat, editovat a smazat sezónu a v rámci sezóny stanovit cenu objektu či jeho částí.

Sekretariát může provádět operace příslušící *návštěvníkovi systému* a navíc:

- zobrazit seznam provedených objednávek, kde může odfiltrovat nevyřízené objednávky;
- potvrzovat či zamítat objednávky;
- zobrazit kontaktní informace o objektu nebo majiteli objektu, ke kterému se objednávka vztahuje;
- přidat nový objekt do systému;
- zobrazit seznam všech objektů v systému a zcela je editovat;
- deaktivovat objekt, tzn. vyhodit ho do koše objektů;
- zobrazit seznam objektů v koši a aktivovat objekty z koše;
- zobrazit seznam všech *registrovaných uživatelů* a *majitelů objektů*, editovat informace o těchto uživatelích včetně nastavení jejich role.

Administrátor může provádět operace příslušící *sekretariátu* a navíc:

- zobrazit seznam všech uživatelů systému a editovat jejich údaje včetně nastavení role;
- deaktivovat uživatele, tzn. vyhodit ho do koše uživatelů;
- prohlížet koš uživatelů a aktivovat uživatele z koše;
- zobrazit seznam typů objektů a aktivovat či deaktivovat jejich zobrazení v nabídce při vytváření objektu;
- přidat typ objektu;
- zobrazit seznam atributů u daného typu objektu, přidávat, editovat a mazat atributy;
- zobrazit seznam oblastí, přidávat, editovat a mazat oblasti;
- zobrazit seznam jazykových mutací, přidávat, editovat a mazat jazykovou mutaci;
- zobrazit seznam překladů šablon a provádět překlady jednotlivých řetězců.

2.2.4 Nefunkční požadavky

Na základě konzultace s vedoucím bakalářské práce a s osobou pohybující se v oblasti ubytování, byly stanoveny následující nefunkční požadavky na vznikající systém:

- Systém bude vytvořen v jazyce Ruby ve frameworku Ruby on Rails.
- Systém poběží na webovém serveru WEBrick s podporou Ruby a Ruby on Rails.
- Systém bude využívat systém řízení báze dat MySQL.
- Systém bude implementován jako webová aplikace.
- K systému se bude přistupovat prostřednictvím sítě Internet a protokolu HTTP.
- Prezentační vrstva systému bude implementována pomocí značkovacího jazyka HTML a kaskádových stylů CSS, pro práci se systémem je tedy třeba webový prohlížeč.
- Požadavky na webový prohlížeč jsou podpora CSS a podpora JavaScriptu, čemuž vyhovuje každý z obvykle užívaných prohlížečů (Firefox, Internet Explorer, Opera, Safari).
- Očekávaná zátěž systému je do 2000 návštěv za den a do 20 uživatelů, kteří systém využívají v jeden moment.
- Očekávaný počet registrovaných uživatelů systému je do 10 000 uživatelů.
- Očekávaný počet objektů v systému je do 1000 objektů.

2.2.5 Specifikace případů užití

Na základě funkčních a nefunkčních požadavků byl vytvořen use-case diagram specifikující případy užití systému. Viz Příloha 1.

Specifikace vybraného detailu případu užití s alternativními kroky:

Případ užití: Objednání objektu
ID: UC11
Účastníci: Registrovaný uživatel
Vstupní podmínky: Registrovaný uživatel je přihlášen do systému
Tok událostí: 1. Případ užití začíná volbou „Objednat ubytování“ u určitého objektu. 2. Systém zobrazí Registrovanému uživateli formulář s předvyplněným číslem objektu, u kterého zákazník zvolil „Objednat ubytování“. 3. Uživatel vyplní čísla objektů o které má zájem pro případ, že objekt, který vybral jako prioritní, nebude k dispozici. 4. Uživatel vyplní od kdy do kdy tento objekt objednává, pro kolik osob, případně další doplňující informace. 5. KDYŽ se časový úsek objednávky nekryje s jinou potvrzenou objednávkou v systému: 5.1 Systém spočítá odhadovanou cenu pobytu. 5.1 Systém zobrazí shrnutí objednávky s odhadovanou cenou pobytu. 5.2 KDYŽ uživatel potvrdí shrnutí objednávky: 5.2.1 Systém uloží objednávku a oznámí Zákazníkovi úspěšné dokončení objednávky. 5.2.2 Případ užití končí. 5.3 Systém znova zobrazí formulář objednávky a vyzve k případným korekcím. 6. Systém vypíše hlášení, že zvolený objekt je ve vybraném termínu obsazen a vyzve k výběru jiného termínu.
Následné podmínky: Sekretariát přijímá objednávku a kontaktuje majitele objektu pro její potvrzení.
Alternativní tok 1: 1. Registrovaný uživatel může kdykoli opustit proces objednávky.
Následné podmínky: Dojde ke ztrátě informací zadaných mezi zahájením a opuštěním procesu objednávky.
Alternativní tok 2: 1. Registrovaný uživatel může kdykoli opustit systém.
Následné podmínky: Dojde ke ztrátě informací zadaných mezi zahájením objednávky a opuštěním systému.

2.2.6 Entiti Relationship Diagram

Na základě specifikace případů užití byl sestaven datový model, viz Příloha 2.

Vzhledem k tomu, že systém je implementován v rámci bakalářské práce a nebude nasazen do reálného provozu, byl v návrhu datového modelu kladen maximální důraz na flexibilitu a robustnost řešení, které bude lehce přizpůsobitelné i pro odlišné požadavky. Cílem flexibilního řešení je nejen to, že požadavky na systém se mohou v průběhu jeho hypotetického provozu změnit. Autor věří, že systém může být skutečně reálně využit, což s sebou jistě jiné než modelové požadavky přinese. Ve zcela ideálním případě by pak mohl být z části návrhovým vzorem, pro systémy řešící rezervaci a ubytování.

Hlavním bodem datového modelu je entita **nodes**, která reprezentuje objekt či jeho část, kterou je možné objednat pro ubytování. Objekty je tak možné hierarchicky strukturovat, například na podlaží, pokoje a lůžka, pomocí relace many-to-one, kdy na rodičovský objekt odkazují jeho potomci.

Pro zajištění flexibility systému se na entitu nodes váže entita **node_types**, která definuje typ jednotlivých objektů a jejich částí, např. typ penzion, typ chata, typ pokoj apod.

Každý typ objektu (node_types) má své specifické atributy definované entitou **attributes**. Např. typ *penzion* může mít jako atribut počet pokojů, dostupnost parkoviště, informaci o stravování, typ *pokoj* potom počet lůžek a vybavení.

Hodnoty jednotlivých atributů pro dané objekty a jejich části nese entita **node_texts**.

Ke každému objektu či jeho části lze definovat sezóny, viz. entita **seasons**, během kterých lze pro daný objekt či jeho část předefinovat jeho defaultní cenu, viz. entita **prices**.

Dále se k objektům vážou fotografie viz. entita **photos**.

Oblasti, ve kterých objekt leží a podle kterých lze objekty filtrovat v katalogu, jsou definovány entitou **areas**.

Podstatnou je entita **users**, která nese informace o každém uživateli systému.

Na entitu users se vážou entity **roles** a **permissions,** pomocí kterých jsou jednotlivým uživatelům definovány role. Jednotlivým rolím jsou definovány oprávnění, podle kterých mohou uživatelé patřící do dané role se systémem pracovat.

Jednotliví uživatelé mohou objekty zařazovat do seznamu svých oblíbených objektů, což je realizováno entitou **favs**, a mohou hodnotit objekty, což je realizováno entitou **ratings**.

Uživatelé, kteří mají roli s patřičným oprávněním, objednávají objekty. Objednávky objektů jsou realizovány entitou **orders** a **ordered_nodes**. Entita ordered_nodes slouží k realizaci případu, kdy je objednán objekt a k němu alternativní další objekty pro případ, že první v pořadí nebude k dispozici.

Entita **supported_locales** je seznamem podporovaných lokalizací a přes relace s entitami lokalizačního pluginu (globalize_country_id, globalize_languages_id) se skrze ní přiřazují *měny* k cenám jednotlivých objektů.

3 Implementace

Implementace přímo vychází z provedené analýzy a návrhu systému. V implementaci byly dodržovány konvence frameworku Ruby on Rails.

Fyzický datový model systému reprezentovaný relační databází je odvozen z logického modelu viz kapitola Entiti Relationship Diagram, struktura tabulek relační databáze vychází z entit.

Vzhledem ke konvencím frameworku jsou ve zdrojových kódech použity anglické názvy pro proměnné, metody, modely, controllery atd. Modely a jejich názvy korespondují s názvy tabulek v relační databázi, mimo tabulek zajišťujících vazbu *many-to-many*, kterou Ruby on Rails řeší interně.

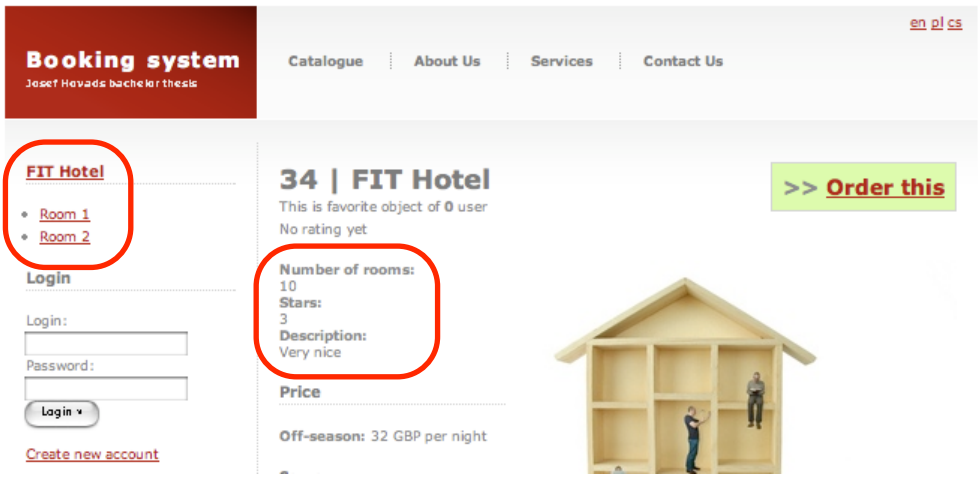
Aplikace byla implementována s jednoduchým uživatelským rozhraním, umožňujícím snadnou modifikaci (vyplývající ze struktury frameworku), pro případné nasazení v konkrétním vizuálním stylu užívaném agenturou provozující systém.

3.1 Objekty

V datovém modelu je možnost strukturovat objekty na menší části (např. *penzion* na *pokoje*). Pro zobrazení každého objektu je tedy zjišťována celá struktura jeho částí. Konkrétně jde o vyhledání nejvyššího nadřazeného objektu, poté o vyhledání všech jeho *potomků*, resp. částí. Při zobrazení uživateli je pak vypsána celá struktura částí objektu (viz obr. 3) a zvýrazněna část, jejíž detaily jsou právě zobrazeny.

Dále datový model umožňuje definici libovolných typů objektů a částí objektu a jejich atributů náležících k těmto typům. Samotné výpisy informací o objektech pak probíhají následujícím způsobem:

- Je vyhledán typ objektu, o němž jsou získávány informace.
- Je vyhledána *množina atributů* náležící danému typu objektu.
- Je vyhledána *množina hodnot atributů* náležící danému objektu.
- Dojte k synchronizaci *množiny atributů* a *množiny hodnot atributů*, kdy jsou jednotlivým atributům přiřazeny hodnoty.
- Dojde k výpisu ve formátu *název atributu: hodnota atributu*, viz obr. 3.



Obr. 3

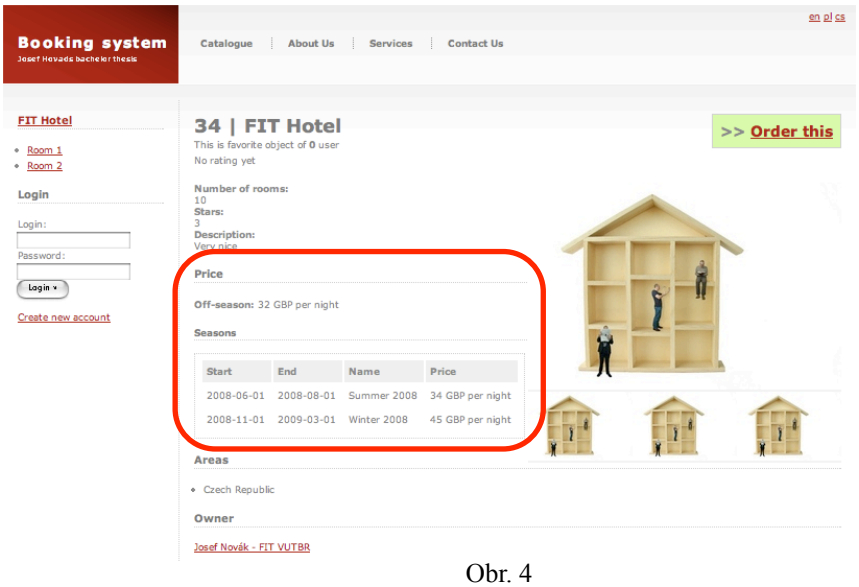
3.2 Sezóny a ceny

Sezóna je období, kdy je obecně zvýšená poptávka po ubytování určitého typu (např. v ČR ubytování v chatách v období na přelomu roku), což logicky vede k tomu, že provozovatelé zvýší pro toto období ceny svých objektů. Jedná se o zcela základní požadavek na rezervační a ubytovací systém.

Vedle standardní ceny objektu, definované pro každý objekt, je tedy vypsána tabulka obsahující případnou právě probíhající sezónu a následující plánované sezóny, viz obr. 4. Sezóny jsou definovány vždy pro celý objekt, nikoli pro jeho části. Při zobrazení části objektu je tedy nejprve nalezen jeho nejvýše nadřazený objekt, pro něj jsou vyhledány sezóny a ty jsou vypsány.

Jednotlivé ceny, které se vážou k sezónám, se však v dané sezóně definují i k jednotlivým částem objektů. Výpis tabulky sezón a cen na stránce s informacemi o objektu či jeho části tedy probíhá následovně:

- Pokud realizován výpis pro část objektu, je vyhledán jeho nejvýše nadřazený objekt.
- Pro objekt je vyhledána množina sezón, zahrnující probíhající sezónu a následující sezóny.
- Pro každou sezónu je hledána cena, která náleží do dané sezóny a k objektu či části objektu, u kterého je tabulka sezón a cen vypisována.
- Pokud je nalezena cena, je vypsána v tabulce sezón a cen, jinak je na stejné místo vypsána defaultní cena pro daný objekt či jeho část.



Obr. 4

3.3 Objednávky

Každá nově provedená objednávka je označena stavem *receved*, neboli *přijatá*. Sekretariát nebo majitel objektu, ke kterému objednávka náleží, pak může její stav změnit na *confirmed* nebo *canceled*, tedy *potvrzena* nebo *zamítnuta*. V každé objednávce je však možnost vybrat alternativní objekty pro případ, že ubytování v prvním vybraném nebude možné. Pokud je tedy objednávka zamítnuta a v řadě objektů zbývají nějaké alternativní, je nastaven příznak k následujícímu objektu a objednávka přesunuta do stavu *receved*, kdy opět čeká na potvrzení či zamítnutí majitelem objektu či sekretariátem.

3.4 Autentifikace a autorizace

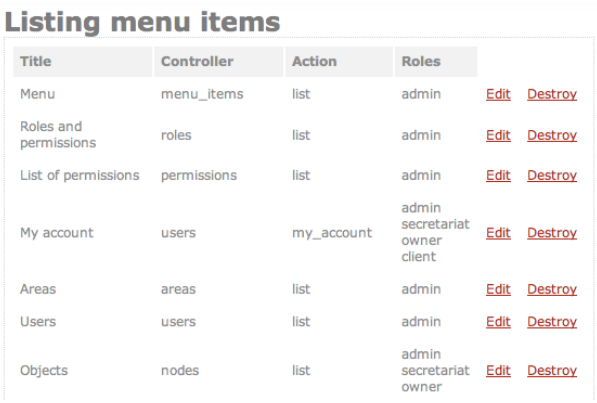
Autentifikace je v aplikaci zajištěna prostřednictvím generátoru login_generator [6], implementace autorizace navazuje na autentifikaci a vychází z datového modelu (entity *roles* a *permissions*).

Systém oprávnění přiřazených jednotlivým rolím uživatelů je v databázi přednastaven podle specifikace případů užití, nicméně administrátor systému má přednastavenou možnost zcela libovolně modifikovat seznam oprávnění (který ovšem musí korespondovat s akcemi jednotlivých controllerů), a přiřazení jednotlivých oprávnění k rolím, viz obr. 5.

Obr. 5

3.5 Systém menu

Systém menu je svázán s uživatelskými rolemi. Administrátor systému má možnost definovat jednotlivé položky uživatelského menu, které se zobrazují definovaným rolím uživatelů, viz obr. 6. V relační databázi je systém menu definován tabulkou *menu_items*.



Obr. 6

3.6 Nastavení systému

Pro zvýšení flexibility systému byla v relační databázi vytvořena tabulka *variables*, definující nastavení systému. Její obsah je administrovatelný prostřednictvím systému. Momentálně jsou definovány nastavení pro definici toho, které uživatelské role korespondují s *registrovaným uživatelem* systému a které s *majitelem objektu* a nastavení délky intervalu pro zobrazení tabulky sezón na prezentační stránce objektu. V případě dalšího rozšiřování systému lze předpokládat rozšiřování množiny obdobných nastavení.

3.7 Pluginy

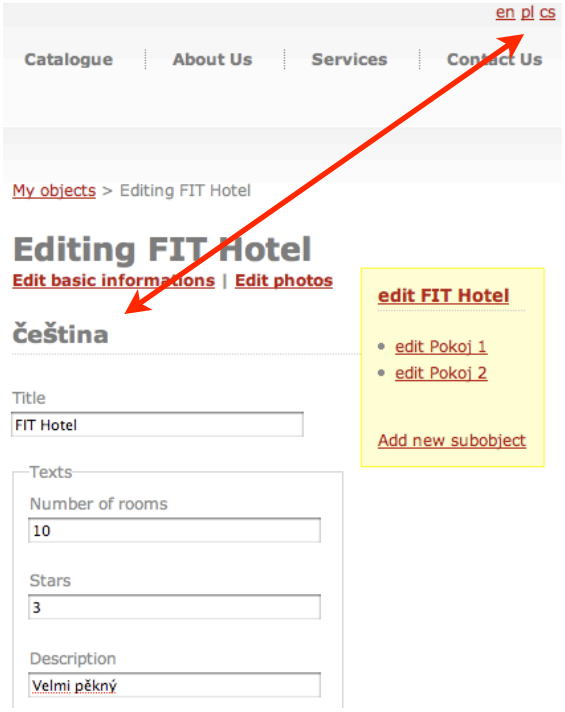
Pro určité části aplikace byly použity následující existující pluginy.

Globalize

Jedná se o plugin poskytující rozhraní pro lokalizaci aplikace. Poskytuje rozhraní pro lokalizaci následujících části:

- Lokalizaci data, čísla a ceny.
- Lokalizaci řetězců uložených v tabulkách relační databáze. Pro vytvoření překladu stačí editovat patřičnou položku v jazyce, do kterého má být přeložena. Pro výpis dat jsou pak voláním metody *find* vráceny již přeložené řetězce.
- Lokalizaci řetězců v šablonách prezentační vrstvy a řetězců chybových hlášení generovaných frameworkem. Text šablony, jehož překlad je žádán, je volán s metodou *t*, která zajistí nalezení patřičného ekvivalentu v tabulce překladů. Např. `"Will be translated".t`.

Více viz [7], obr. 7.



Obr. 7

File Column Plugin

Usnadňuje práci se soubory a obrázky. V aplikaci je použit pro usnadnění práce s fotografiemi, které je možné připojovat k objektům a jejich částem, viz obr. 8. Více viz [8].

Obr. 8

Redbox

Jedná se o obdobu známých JavaScript knihoven Lightbox a Thickbox implementovanou pro snadné použití v Ruby on Rails. V aplikaci je tento plugin využit pro uživatelsky přívětivé zobrazování plného rozlišení fotografií objektů. Více viz [9].

4 Závěr

Závěrečná kapitola obsahuje zhodnocení dosažených výsledků. Hodnotí zvláště části významné přínosem autora této práce. Následuje zhodnocení z pohledu dalšího vývoje projektu. Na závěr jsou diskutovány výhody a nevýhody použití frameworku Ruby on Rails.

4.1 Zhodnocení dosažených výsledků

Autor práce se seznámil s programovacím jazykem Ruby a s moderními přístupy objektového návrhu pro webové aplikace jako jsou návrhové vzory a frameworky. Především potom důkladně s frameworkem Ruby on Rails. Dále se autor seznámil s požadavky kladenými na reservační a ubytovací systémy a prostudoval již existující řešení. Na základě toho navrhl vícejazyčnou webovou aplikaci pro katalogizaci, administraci a reservaci ubytovacích zařízení s použitím modelovacích prostředků UML a návrh implementoval ve frameworku Ruby on Rails.

Při návrhu systému kladl autor důraz především na flexibilitu výsledného systému. Systém je tak možné použít k nabídce ubytování v jakémkoli typu ubytovacího zařízení. Dále systém umožňuje strukturování objektů, v nichž je ubytování nabízeno, na jakékoli menší části (patra, pokoje, lůžka apod.). Systém je také možné modifikovat z pohledu uživatelů, rolí a oprávnění, přičemž lze role a k ním náležící oprávnění jakkoli měnit. Za zmínku stojí také flexibilita *menu*, ve kterém lze opět volně předefinovat položky pro jednotlivé role. Dalo by se říci, že po patřičném nastavení může systém používat jakákoli agentura, která zprostředkovává ubytování.

4.2 Porovnání výsledného řešení s již existujícími

Hlavním benefitem vytvořeného systému je, jak již bylo zmíněno, **flexibilita**. Primárním zdrojem inspirace pro vzniklý systém byl systém agentury *Limba s.r.o*.

V porovnání s tímto a ostatními existujícími systémy má systém vytvořený v rámci této bakalářské práce především možnost rozčleňování objektů na části. Je tak např. na rozdíl od systému *Limba* použitelnější například pro nabídku ubytování v objektech typu *hotel* nebo *penzion*, kde zákazník objednává konkrétní pokoj a stejně dobře použitelný např. pro nabídku chat a to i v rámci jednoho katalogu. Např. systém *Limba* má na druhou stranu v porovnání s vytvořeným systémem více funkcí, např. automatický výpis zajímavostí v okolí nabízeného objektu. Nicméně tyto dodatečné funkce, které přímo nesouvisí s hlavním cílem systému, kterým je nabídkový katalog a možnost rezervace, lze snadno implementovat i do systému vytvořeného v rámci této práce.

Obdobně jako existující řešení nabízí vytvořený systém správu sezón, geografických oblastí, objednávek apod. Přihlášenému uživateli pak nabízí možnost zařazovat objekty z katalogu do seznamu jeho oblíbených objektů. Dále pak objekty hodnotit známkou, přičemž průměrné hodnocení je zobrazeno u objektu v katalogu.

4.3 Možnosti dalšího vývoje projektu

V rámci bakalářské práce byl systém vytvářen pro fiktivní agenturu nabízející ubytování. Jeho reálné nasazení však určitě možné je, nicméně přineslo by s sebou pravděpodobně několik specifických požadavků. Díky flexibilnímu návrhu systému, ale také díky tomu, že je systém postavený na frameworku, však bude další rozšiřování systému velmi snadné.

Očekávaná jsou především rozšíření dodatečných funkcí, které budou např. automaticky generovat upřesňující informace k objektům podle jeho polohy, budou automaticky zobrazovat umístění objektu na mapě, přepočítávat ceny do ostatních měn podle momentálního kursu a podobně.

Dále je možné očekávat rozšíření funkcí pro administrátory, jako např. hromadné rozesílání informačních e-mailů, různé formy statistik apod.

4.4 Zhodnocení užití frameworku Ruby on Rails

4.4.1 Výhody

Výhody, které použití Ruby on Rails přineslo, vychází v první řadě z podstaty frameworku postaveném na návrhovém vzoru MVC. Díky tomu, že framework jasně rozděluje jednotlivé aplikační vrstvy, je kód aplikace stále přehledný. To je benefitem nejen pro vývojáře, který na projektu aktuálně pracuje, ale také pro jeho kolegy či budoucí nástupce.

Ruby on Rails je však postaveno na filozofii, která u ostatních frameworků tohoto typu není zcela obvyklá. Již zmíněný princip *konvence má přednost před konfigurací*, najdeme na každém kroku. Ruby on Rails se nesnaží být naprosto flexibilním a nesnaží se řešit všechny případy, které mohou nastat. Naopak vede vývojáře po cestě konvencí, nutné je definovat pouze nestandardní situace. Musím říci, že přes skeptický přístup, kdy jsem se na Ruby on Rails díval jako na „dobrým marketingem vytvořenou bublinu", mi byla práce s tímto frameworkem velmi příjemná a svůj pohled na tento framework jsem v průběhu práce změnil.

Ruby on Rails přináší výhody z hlediska technologického (MVC, ActiveRecord apod.) i ekonomického (standardizace a urychlení vývoje).

4.4.2 Nevýhody

Mezi relativní nevýhody v použití Ruby on Rails by se dala považovat především zatím malá podpora u webhostingových služeb a v České republice obecně zatím malá komunita vývojářů (což nemusí být nutně nevýhoda, např. vzhledem k tomu kolik vývojářů se považuje za „profesionály" v PHP). Často kritizované jsou také požadavky na výkon hardware, kdy aplikace napsané v Ruby on Rails jsou obvykle poněkud náročnější. Nicméně i na této skutečnosti se při vývoji nových verzí Ruby a Ruby on Rails pracuje. Celkově tyto nevýhody považuji za méně podstatné než již zmíněné výhody.

Literatura

[1] THOMAS, D.; HANSSON, D. Agile Web Development with Rails Second Edition. 2006. ISBN 978-0-9776166-3-3.

[2] FITZGERALD, M. Ruby Pocket Reference. 1st ed. 2007. ISBN 978-0-596-51481-5.

[3] HOLZNER, S. Začínáme programovat v Ruby on Rails. 1st ed. 2007. ISBN 978-80-251-1630-2.

[4] DVOŘÁK, M. Návrhové vzory (design patterns) - diplomová práce M. Dvořáka. [online]. , 2008-02-02 [cit. 2008-05-09]. Available from www: <http://objekty.vse.cz/Objekty/Vzory>

[5] About Ruby. [online]. [cit. 2008-05-09].
Available from www: <http://www.ruby-lang.org/en/about>

[6] Login Generator. [online]. , 2006-07-12 [cit. 2008-05-09].
Available from www: <http://rubyforge.org/projects/login/>

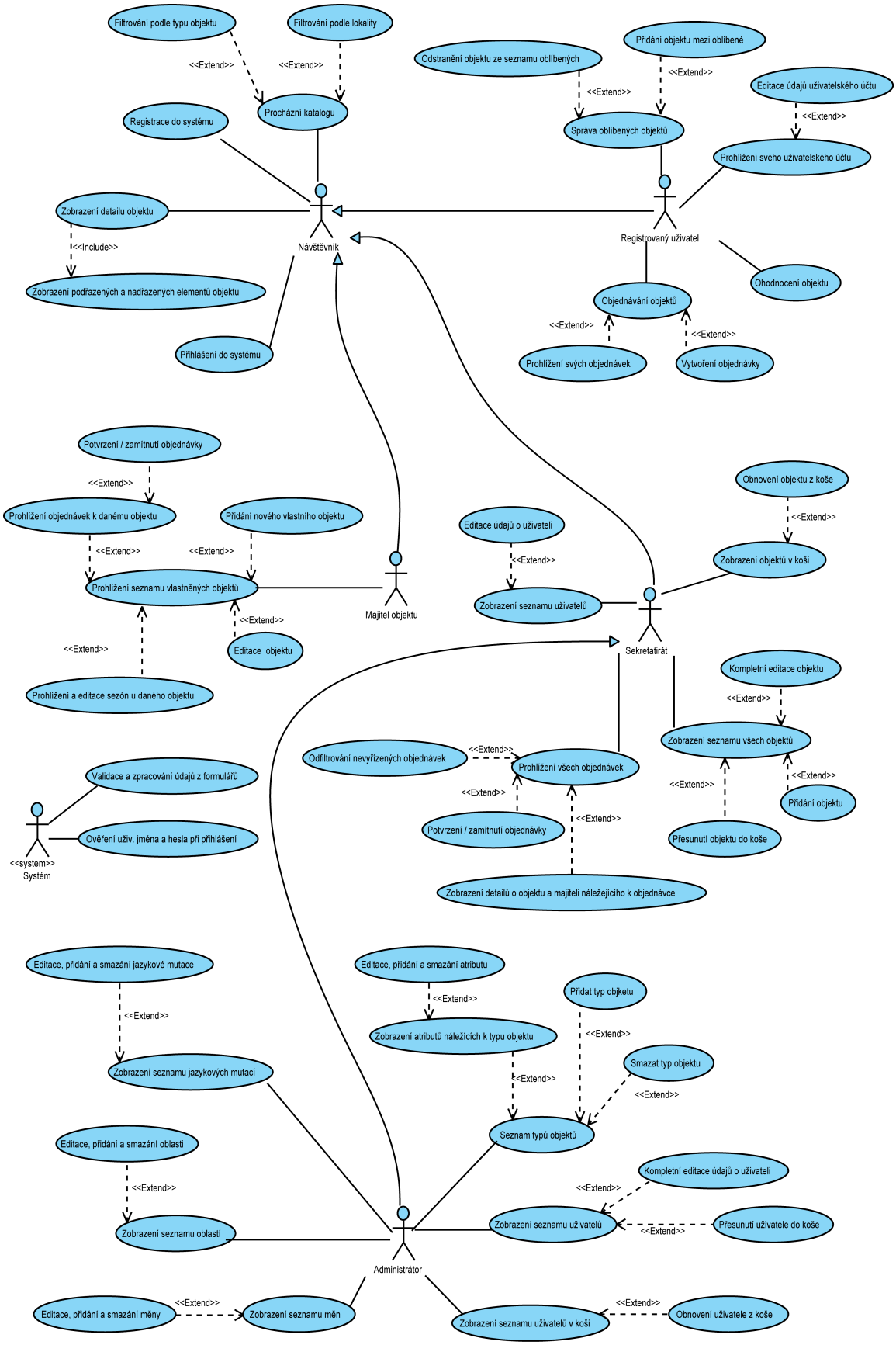
[7] Globalize. [online]. [cit. 2008-05-09]. Available from www: <http://www.globalize-rails.org>

[8] KANTHAK, S. FileColumn - easy handling of file uploads in Rails. [online]. [cit. 2008-05-09].
Available from www: <http://www.kanthak.net/opensource/file_column>

[9] AMBROSE, C. Red Box. [online]. [cit. 2008-05-09].
Available from www: <http://www.craigambrose.com/projects/redbox>

Seznam příloh

Příloha 1. Use-case diagram



Příloha 2. Entiti Relationship Diagram