

Systém pro evidenci hospitací v Ruby on Rails

Rozšiřte prototyp aplikace pro evidenci hospitací (<http://kvalitavyuky.felk.cvut.cz/>) tak, aby jej bylo možné nasadit do reálného provozu na ČVUT FEL. Systém implementujte na platformě Ruby on Rail. Vývoj provádějte iterativním způsobem a postupně zapracovávávejte požadavky zadavatele. Celý vývoj řádně dokumentujte a výsledky své práce otestujte. Funkčnost systému demonstруйте na hospitacích, které budou probíhat v programu STM v LS 2011/2012.

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Systém pro evidenci hospitací v Ruby on Rails

Tomáš Turek

Vedoucí práce: Ing. Martin Komárek

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

14. května 2012

Poděkování

Zde můžete napsat své poděkování, pokud chcete a máte komu děkovat.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

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

V České Lípě dne 5.2.2012

.....

Abstract

The purpose of this work is to design and develop a system for evidence inspections of classes. This application will be used at the Faculty of Electrical Engineering. Reason to create the system is to relieve the administrative burden that increases with increase in the number of observations. The application is based on the existing processes of management inspections for the study program STM. The application is created on the platform of Ruby on Rails and builds on a prototype application. The real development of the application is headed by an iterative method of software development. The resulting application is deployed at <https://kvalitavyuky.felk.cvut.cz>. I have demonstrated the functionality of the on-going observation in the summer semester 2011/2012.

Abstrakt

Účelem této práce je navrhnout a vytvořit systém pro evidenci hospitací. Tato aplikace bude používána na Fakultě elektrotechnické. Důvodem vzniku systému je ulehčit administrativní zátěž, která se zvyšuje s nárůstem počtu hospitací. Aplikace je založena na již existujících procesech správy hospitací pro studijní obor STM. Aplikace je vytvořena na platformě Ruby on Rails a navazuje na prototyp aplikace. Samotný vývoj aplikace je veden iterativní metodou vývoje softwaru. Výsledná práce je nasazená na adrese <https://kvalitavyuky.felk.cvut.cz>. Funkčnost jsem demonstroval na probíhajících hospitacích v letním semestru 2011/2012.

Obsah

1	Úvod	1
2	Popis problému, specifikace cíle	3
2.1	Motivace	3
2.2	Cíle práce	3
2.3	Rešerše	3
2.3.1	Prototyp	4
2.3.2	Postupy pro kontrolu kvality výuky	4
2.3.2.1	Administrátor hospitací	4
2.3.2.2	Plánování hospitací	4
2.3.2.3	Typy hospitací	4
2.3.2.4	Provedení hospitace	5
2.3.2.5	Hodnocení výuky	5
3	Analýza	7
3.1	Požadavky	7
3.1.1	Obecné požadavky	7
3.1.2	Funkční požadavky	7
3.2	Uživatelské role	8
3.2.1	Nepřihlášený uživatel	9
3.2.2	Přihlášený uživatel	9
3.2.3	Hospitovaný	9
3.2.4	Hospitující	9
3.2.5	Administrátor hospitací	9
3.2.6	Administrátor	9
3.3	Doménový model	10
3.3.1	Domény z KOSapi	10
3.3.2	Domény aplikace	10
3.4	Životní cyklus hospitace	12
3.4.1	Vytvoření	12
3.4.2	Naplánování	12
3.4.3	Hodnocení	12
3.4.4	Ukončená	12

4	Návrh	15
4.1	Technologie a služby	15
4.1.1	Ruby on Rails	15
4.1.1.1	Balíčkový systém RubyGems	15
4.1.2	KOSapi	15
4.1.3	FELid	16
4.1.4	Aplikační server	16
4.1.5	Databáze	16
4.2	Architektura	16
4.2.1	MVC	16
4.2.1.1	Models	17
4.2.1.2	Views	17
4.2.1.3	Controllers	17
4.2.2	DRY	17
4.2.3	CoC	17
4.2.4	REST	17
4.3	Struktura aplikace	18
5	Realizace	21
5.1	První iterace	21
5.1.1	Zadání	21
5.1.2	Postup	21
5.1.2.1	Datová vrstva	21
5.1.2.2	Autentizace	22
5.1.2.3	Autorizace	22
5.1.2.4	Role	22
5.1.2.5	Uživatelské prostředí	23
5.1.3	Výstup	23
5.2	Druhá iterace	23
5.2.1	Zadání	23
5.2.2	Postup	23
5.2.2.1	Datová vrstva	23
5.2.2.2	Hodnotící formuláře	24
5.2.3	Výstup	26
5.3	Třetí iterace	26
5.3.1	Zadání	26
5.3.2	Postup	26
5.3.2.1	Nasazení aplikace	26
5.3.2.2	FELid	26
5.3.2.3	Záloha databáze	27
5.3.3	Výstup	27
5.4	Čtvrtá iterace	27
5.4.1	Zadání	27
5.4.2	Postup	28
5.4.2.1	Autentizace	28
5.4.2.2	Emaily	28

5.4.2.3	Generátor obsahu emailových zpráv	28
5.4.2.4	Odesílání emailů	30
5.4.3	Výstup	30
6	Testování	31
6.1	Automatické testování	31
6.1.1	Testovací data	31
6.1.2	Unit testing	31
6.1.3	Functional tests	31
6.1.4	Integration testing	32
6.2	Manuální testování	32
7	Závěr	33
7.1	Možnosti v pokračování práce	33
A	Seznam použitých zkratk	37
B	UML diagramy	39
B.1	Use cases	39

Seznam obrázků

3.1	Aktéři	8
3.2	Doménový model	11
3.3	Život hospitace	13
4.1	Struktura aplikace	19
5.1	Doménový model dynamických formulářů	24
5.2	Diagram nasazení	27
5.3	Struktura značky	29
B.1	Use case - nepřihlášený uživatel	39
B.2	Use case - přihlášený uživatel	40
B.3	Use case - hospitalovaný	40
B.4	Use case - hospitalující	41
B.5	Use case - administrátor	41
B.6	Use case - administrátor hospitalací	42

Seznam tabulek

5.1	Reprezentace rolí v bitové masce	22
5.2	Popis atributů modelu PeopleRelated	24
5.3	Seznam typů elementů	25

Kapitola 1

Úvod

Na FEL ČVUT byl zaveden pro studijní program STM (Softwarové technologie a management) systém pro ověřování kvality výuky. Cílem tohoto systému je zkvalitnit vyučované předměty. Jedním ze zdrojů informací jsou kontrolní návštěvy ve výukách¹. Účelem těchto návštěv je získání komplexního obrazu o kvalitě výuky pro garanty jednotlivých předmětů a zároveň slouží pro pedagogy jako zpětná vazba z přednášek a cvičení.

Mým cílem mé práce je navrhnout a vytvořit informační systém pro správu hospitací, který zjednoduší a zrychlí administrativu, ke které se doposud používala e-mailová komunikace a www stránky Rady programu [24]. Systém je postaven na moderním frameworku Ruby on Rails, který je určen pro vývoj moderních webových aplikací. Systém využívá dvě fakultní aplikace FELid a KOSapi.

¹zkráceně hospitace

Kapitola 2

Popis problému, specifikace cíle

2.1 Motivace

Jak jsem uvedl v úvodu, tak v současné době probíhá jakákoliv administrativní činnost okolo hospitací převážně pomocí emailové komunikace. Kterou zajišťuje garantem studijního oboru přidělený administrátor kontroly výuky, který je dále uváděn jako administrátor, nebo administrátor hospitací.

Jeho úkolem je starat se o plánování hospitací a vystavování dokumentů na stránkách rady studijního programu [24]. Při naplňování hospitace musí administrátor ručně obeslat emailem informaci o naplňované hospitaci všem zainteresovaným osobám. To jsou hospitovaní, hospitující, přednášející a garanti příslušného předmětu. Po provedení hospitace je potřeba shromáždit a vystavit veškeré dokumenty na webových stránkách Rady programu. Administrátor musí hlídat tok dokumentů a rozesílat vzniklé dokumenty mezi účastníky hospitace.

Tento systém sice funguje, ale je administrativně a časově náročný při zvyšujícím se počtu hospitací jak pro administrátora hospitací, který se stará o komunikaci mezi účastníky, tak i pro zúčastněné strany hospitace. Proto byl podán požadavek na vytvoření systému pro evidenci hospitací, který zautomatizuje vnitřní procesy pro správu hospitací.

2.2 Cíle práce

Hlavním cílem mé práce je rozšířit prototyp aplikace pro evidenci hospitací, tak aby bylo možné ji nasadit do reálného provozu na FEL ČVUT. V průběhu letního semestru 2011/2012 se bude postupně demonstrovat její funkčnost na realizovaných hospitacích v daném semestru.

2.3 Rešerše

Hospitace jak už jsem nastínil v motivaci je zavedený vnitřní proces kontroly kvality výuky na ČVUT FEL, proto čerpám informace pro tvorbu této práce z dvou hlavních zdrojů. Prvním zdrojem je dokument Postupy pro kontrolu kvality výuky [22], který definuje jak se

mají hospitace provádět. A druhým zdrojem je prototyp aplikace z rozpracované bakalářské práce Návrh a implementace systému pro správu hospitací [25].

2.3.1 Prototyp

Jako referenční řešení práce jsem obdržel prototyp aplikace napsaný v Ruby on Rails. Ten se skládá z dokumentační části a z rozpracované webové aplikace. Dokumentační část prototypu obsahuje analýzu i návrh aplikace. Hlavně na začátku mi velmi pomohl pochopit problematiku hospitací a obsahovala velmi užitečné informace pro samotný vývoj aplikace. Horší to bylo se samotnou aplikací. Verze programu kterou jsou získal byla velmi rozpracovaná jelikož se mi ji nepodařilo zprovoznit tak, aby byla funkční. Po hlubším zkoumání a upravování zdrojového kódu jsem zjistil, že aplikace umí plánovat hospitace. Dále byli vytvořeny html šablony pro hodnotící formuláře. Prototyp také obsahuje knihovnu pro komunikaci s aplikací KOSapi¹.

Důvod proč navazuji na rozpracovanou práci je ten že autor se rozhodl kompletně přejít na jinou technologii než je Ruby on Rails. Nakonec po hlubší analýze prototypu programu jsem rozhodl převzít z prototypu aplikace pouze připojení ke KOSapi a zbytek udělat od základů nový.

2.3.2 Postupy pro kontrolu kvality výuky

V této části kapitoly popisují hlavní procesy při vykonávání hospitací. Tyto informace čerpám z již zmíněného dokumentu Postupy pro kontrolu kvality výuky [22], kde jsou obsaženy veškeré potřebné informace o průběhu hospitací.

2.3.2.1 Administrátor hospitací

Administrátora hospitací je jedna pro studijní program, kterou vybírá garant studijního oboru. Tato osoba se stará o hospitace pro studijní obor ke kterému je přiřazen. Jeho hlavními úkoly jsou plánování a řízení hospitací.

2.3.2.2 Plánování hospitací

Pro naplánování hospitace musí administrátor hospitací definovat předmět, paralelku a datum hospitace. Hospitace může probíhat jak na přednášky tak i na cvičení. Do plánování patří i přiřazení hospitujících², kteří provedou kontrolní návštěvu ve výuce.

2.3.2.3 Typy hospitací

Administrátor také určuje typ hospitace. Jsou zavedeny tři druhy a především se rozlišují z hlediska hospitovaného. Jsou to:

¹viz. 4.1.2

²typicky je to dvojce pedagogů jeden zkušený a druhý začínající

Předem ohlášené na konkrétní datum u tohoto typu hospitace jsou veřejné všechny informace o naplánování hospitace. Proto je potřeba, aby administrátor naplánoval hospitaci s předstihem a informoval o tom hospitovaného.

Předem ohlášené bez konkrétního termínu tento typ hospitace na rozdíl od předešlého typu nemá pevně stanovený datum hospitace. Hospitovaný sice ví o naplánovanou hospitaci, ale není zveřejněn datum kdy proběhne.

Předem neohlášené tento typ hospitace se používá jen zřídka a to u problémových předmětů, nebo při vážných stížnostech na vyučujícího. Proto u tohoto typu hospitace je předem informován pouze garant, zástupce garanta a administrátor výuky.

2.3.2.4 Provedení hospitace

Hospitace se vykoná kontrolní návštěvou hospitujících na výuce a sepiší dokument Hodnocení výuky. V něm zdokumentují průběh hodiny a její hodnocení.

2.3.2.5 Hodnocení výuky

Po vykonání hospitace následuje hodnotící fáze. Ta se skládá ze čtyř druhů dokumentů, které zhodnotí proběhlou výuku. Všechny dokumenty musí být předány administrátorovi hospitací. Ten je vystaví na privátní části webových stránek Rady programu [24]. Po sepsání a vystavení všech dokumentů je hospitace ukončená. Dokumenty používané při hodnocení:

- A Hodnocení výuky při hospitaci - ten dokument je písemný výstup z hospitace. Hlavním účelem tohoto dokumentu je slovně popsat průběh výuky. Dokument se skládá z dokumentační části a z hodnotící části. Tento formulář vyplňuje každý hospitující sám za sebe.
- B Slovní hodnocení hospitační návštěvy hospitujícím(mi) - tento dokument sepiše po hospitaci jeden z hospitujících a účelem tohoto dokumentu je slovně zhodnotit výuku.
- C Stanovisko hodnoceného učitele k názorům hospitujícího - Tímto dokumentem se může hospitovaný, garant předmětu a vedoucí katedry vyjádřit k slovnímu hodnocení. Proto tento dokument lze sepsat až po vystavení formuláře B.
- D Závěrečné shrnutí hospitujícím - je poslední dokument a nejdůležitějším dokumentem ze všech. Sepíše ho jeden z hospitujících. Tento dokument slouží jako výstup hodnocení hospitace. Dokument obsahuje klady, zápory, navržená opatření a závěr. Je také jediným veřejným dokumentem a proto je potřeba ho vytavit na volně přístupných stránkách.

Kapitola 3

Analýza

Tato kapitola pojednává o analýze. Výstupem této části jsou funkční, obecné požadavky, doménový model, aktéři a k nim případy užití.

3.1 Požadavky

Požadavky na systém se dělí na dvě sekce: obecné a funkční požadavky. Pro definování těchto požadavků jsem převzal z prototypu aplikace [25] a rozšířil jsem je. Tyto požadavky mi definují návrh aplikace a pro technologie potřebné pro implementaci systému.

3.1.1 Obecné požadavky

Obecné požadavky se netýkají funkčnosti, ale celkového návrhu a použitých technologií.

1. Systém bude postaven na webovém frameworku Ruby on Rails.
2. Systém bude webovou aplikací.
3. Systém bude používat webovou službu KOSapi.
4. Systém bude pro autentizaci používat aplikaci FELid.

3.1.2 Funkční požadavky

Tato sekce se zabývá požadavky na funkčnost systému.

1. Systém umožní průběžně plánovat hospitace.
2. Systém umožní hospitujícímu i hospitalovanému prohlížet hospitace.
3. Systém umožní vystavit závěrečné hodnocení na veřejné části aplikace.
4. Systém umožní hospitalovanému sepsat stanoviska k názorům hospitujícího.
5. Systém umožní hospitujícímu nahrát naskenovaný dokument Hodnocení výuky.

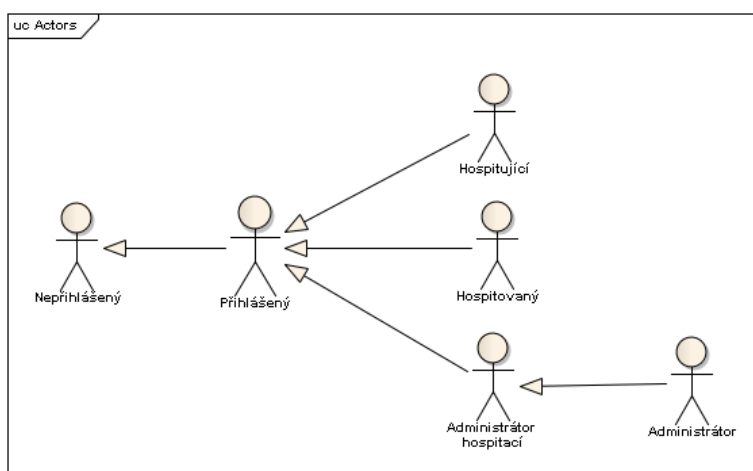
6. Systém umožní hospitujícímu napsat slovní hodnocení z výuky.
7. Systém umožní hospitujícímu napsat závěrečné shrnutí hospitace.
8. Systém bude odesílat informační emailem zprávy při vyplnění hodnotícího dokumentu.
9. Systém umožní vyhledávat předměty z KOSapi.
10. Systém umožní vyhledávat osoby z KOSapi.
11. Systém umožní upravovat strukturu hodnotících formulářů.
12. Systém umožní spravovat emailové šablony.
13. Systém umožní generovat emailové zprávy ze emailových šablon.
14. Systém bude automaticky zálohovat databázi.

3.2 Uživatelské role

V systému je celkem 7 uživatelských rolí. Definoval jsem tři základní uživatelské role, které jsou základem systému: nepřihlášený uživatel, přihlášený uživatel, administrátor hospitací a admin. Další dvě role hospitovaný a hospitující se přidělují v rámci jednotlivých hospitací. Na obrázku 3.1 jsou vidět jednotliví aktéři a jejich zobecnění. V další části této sekce rozeberu jednotlivé role a k nim případy užití.

Use cases neboli případy užití je nástroj pro popsání chování jak by systém měl spolupracovat s koncovým uživatelem. Popisuje všechny způsoby jak uživatel komunikuje se systémem.

Důležitou součástí Use Case modelu jsou také scénáře užití. Tyto scénáře blíže rozvádějí jednotlivé aktivity stakeholderů a reakce modelovaného systému.



Obrázek 3.1: Aktéři

3.2.1 Nepřihlášený uživatel

Nepřihlášený uživatel je role pro hosty naší aplikace. V systému má ze všech rolí nejmenší pravomoc. V tomto stavu je každý uživatel, který se doposud nepřihlásil do systému.

3.2.2 Přihlášený uživatel

Přihlášený uživatel vychází z role nepřihlášeného uživatele. Je to uživatel, který se do systému přihlásil. Jedná se o základní roli pro všechny další role, které ji rozšiřují.

3.2.3 Hospitovaný

Hospitovaný je role pro přihlášeného uživatele v systému. Je přidělena pro každého vyučujícího, který vyučuje předmět, na němž byla naplánovaná hospitace a proběhla.

3.2.4 Hospitující

Hospitující je role pro přihlášeného uživatele v systému. Tato role se přiděluje automaticky z naplánovaných hospitací, nebo ji může přidělit administrátor hospitací.

3.2.5 Administrátor hospitací

Hlavním úkolem této role je plánovat hospitace na předměty a posléze je spravovat.

3.2.6 Administrátor

Administrátor je super uživatel, který má nejvyšší pravomoc v systému. Má přístup ke všem zdrojům aplikace a může aplikaci spravovat.

3.3 Doménový model

Doménový model na obrázku 3.2 reprezentuje entity v systému a jejich vzájemné vztahy.

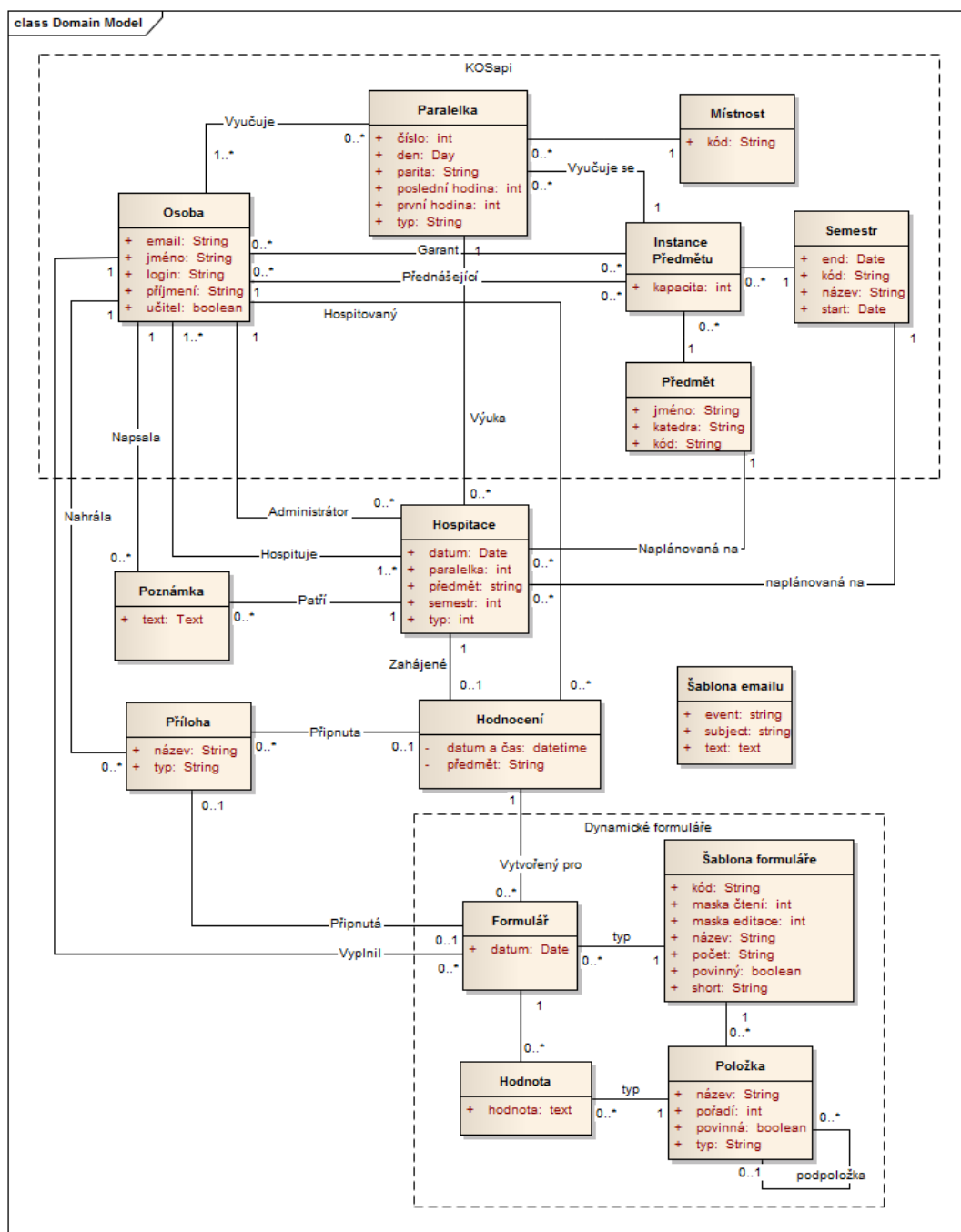
Popis domény jsem pro přehlednost rozdělil podle zdroje na dvě základní skupiny. V první skupině jsou domény, které jsem převzal ze struktury KOSapi a druhou skupinou jsou domény specifické pro moji aplikaci.

3.3.1 Domény z KOSapi

- Osoba - je osoba v KOSu. Každá osoba může být učitelem a studentem.
- Semestr - semestr vyučovaný na FEL.
- Předmět - předměty vyučované na FEL.
- Instance předmětu - jsou instance předmětu vypsány v konkrétním semestru.
- Paralelka - je vypsána rozvrhová paralelka pro instanci předmětu.
- Místnost - místnost na FEL

3.3.2 Domény aplikace

- Hospitace - obsahuje informace o naplánování hospitace.
- Poznámka - je textová poznámka k plánování hospitace.
- Hodnocení - reprezentuje informace z proběhlé hospitace, jsou to informace o datu vykonání hospitace, hospitujícím, předmětu a garantovi.
- Příloha - je připojený datový soubor k hodnocení hospitace.
- Šablona formuláře - šablona pro tvorbu formulářů. Definuje vlastnosti jakým se budou vytvářet hodnotící formuláře.
- Položka - položka reprezentuje jednotlivé segmenty formuláře. Tyto segmenty pak v celku definují strukturu formuláře.
- Formulář - vyplněný hodnotící formulář.
- Hodnota - je hodnota z vyplněného formuláře. Ta se ukládá z položky formuláře.
- Šablona emailu - šablona emailu ze které se budou generovat emailové zprávy.



Obrázek 3.2: Doménový model

3.4 Životní cyklus hospitace

Cílem této části analýzy je popsat životní cyklus, kterým hospitace prochází.

3.4.1 Vytvoření

Životní cyklus hospitace začíná jejím vytvořením. Toto zajišťuje administrátor hospitací, který založí hospitaci a definuje semestr, kdy se má hospitace uskutečnit, a předmět vyučovaný na fakultě. Při vytváření hospitace se určí typ hospitace a tím i její způsob zviditelnění, pro ostatní aktéry v aplikaci.

3.4.2 Naplánování

Při plánování je také hlavním aktérem administrátor hospitace. V této části životního cyklu administrátor určí hospitovanou paralelku předmětu a datum, kdy se hospitace uskuteční.

Administrátor také v této fázi přidělí hospitující z řad pedagogů určených k vykonání hospitace.

3.4.3 Hodnocení

Poté, co proběhla kontrola hospitace, začíná nová fáze, ve které se hodnotí vyučování. Do této fáze už nezasahuje administrátor hospitace, ale přicházejí na scénu dva jiní aktéři: hospitovaný a hospitující.

V první fázi musí hospitující vyplnit, nebo nahrát naskenovaný formulář pro Hodnocení výuky při hospitaci. Tento formulář slouží k dokumentaci průběhu hospitace.

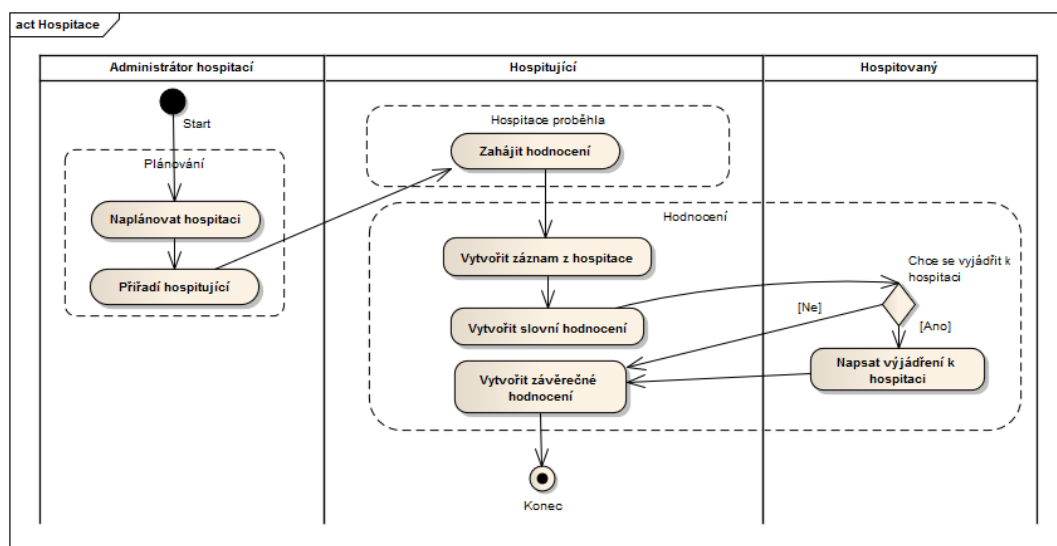
V druhé fázi jeden z hospitujících sepíše slovní hodnocení hospitační návštěvy.

Ve třetí fázi může hospitovaný do dvou dnů vyplnit stanovisko hodnoceného k názorům hospitujícího.

V poslední fázi jeden z hospitujících sepíše poslední formulář Závěrečné shrnutí. Po vyplnění tohoto formuláře se hospitace stává ukončenou a tím končí její životní cyklus.

3.4.4 Ukončená

Po sepsáním posledního hodnotícího dokumentu se hospitace dostane do fáze ukončená.



Obrázek 3.3: Život hospitace

Kapitola 4

Návrh

4.1 Technologie a služby

Tato část popisuje jednotlivé technologie a služby potřebné pro implementaci aplikace.

4.1.1 Ruby on Rails

Ruby on Rails [14], zkráceně Rails, je jedno z implementačních omezení, které se nachází přímo v zadání práce. Jedná se o framework primárně určený pro vývoj webových aplikací napojených na relační databázi. Framework je napsán na skriptovacím interpretovaném programovacím jazyku Ruby [13]. Rails používá návrhový vzor Model-view-controller viz. 4.2.1. Mezi základními myšlenkami a filozofií frameworku patří dva principy. Prvním principem je Convention over Configuration viz. 4.2.3 a druhým je Don't Repeat Yourself viz. 4.2.2.

4.1.1.1 Balíčkovací systém RubyGems

Protože jsou Rails postaveny na jazyku Ruby lze používat balíčkovací systém RubyGems, kde existuje velké množství již existujících řešení, které jsou dostupné jako knihovny¹.

4.1.2 KOSapi

KOSapi je webová služba poskytující aplikační rozhraní v podobě RESTful webové služby 4.2.4. Je určená pro vznik školních aplikací, které potřebují mít přístup k datům souvisejících s výukou. Pro aplikaci používám stabilní verzi API 2.

Z této služby čerpám hlavně data předmětů a osob v KOSu. Pro připojení ke KOSapi používám již existující knihovnu napsanou v Ruby Tomášem Linhartem a Tomášem Jukínem ve školním projektu VyVy [17].

¹v RubyGems se knihovna nazývá Gem

4.1.3 FELid

FELid [7] je globální autentizační a autorizační systém pro webovské aplikace na síti FEL. Poskytuje jednotný a bezpečný způsob přihlášení uživatelů a přenos jejich údajů do různých aplikací na webu. Zároveň podporuje jednorázové přihlášení (tzv. single sign-on). Znamená to, že se uživatel přihlašuje pouze do první použité aplikace a u dalších aplikací už nemusí zadávat svoje přihlašovací údaje.

Tuto službu používám pro autentizaci uživatelů do systému. Abych mohl používat v aplikaci FELid je nutné splnit technické požadavky, které jsou napsány na stránkách FELid [8].

4.1.4 Aplikační server

Pro zprovoznění aplikace do reálného provozu jsem použil webový server Apache HTTP server [20] ve verzi 2. Tento aplikační server jsem zvolil kvůli obecným požadavkům aplikace pro využití FELid a požadavku aby aplikace byla napsaná v Ruby on Rails. Tato verze webového serveru totiž umožňuje instalaci zásuvných modulů Passenger [26] a Shibboleth [23]. Passenger umožňuje nasazení rails aplikací na aplikačním serveru. Druhý modul Shibboleth zprostředkovává single sign-on autentizaci mezi aplikačním serverem a službou FELid.

4.1.5 Databáze

Ruby on Rails poskytuje možnost připojení k různým databázovým systémům prostřednictvím adaptérů. Díky tomu není aplikace závislá na použitém databázovém systému a díky tomu mohu používat pro vývoj a testování aplikace jednoduchý databázový systém SQLite [21], který pro tyto účely bohatě postačuje a není potřeba jej složitě konfigurovat. Pro samotné nasazení aplikace do provozu už používám databázový systém MySQL [19].

4.2 Architektura

V této části popisuji použité architektonické vzory a konvence, které dodržuji při návrhu aplikace.

4.2.1 MVC

MVC (Model-view-controller) [11] je softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent² tak, že modifikace některé z nich má minimální vliv na ostatní. Tento architektonický vzor obsahuje ve svém jádru Ruby on Rails, proto pro implementaci tohoto vzoru vycházím z fungování frameworku.

²models, views a controllers

4.2.1.1 Models

Model reprezentuje informace v aplikaci a pravidla pro práci s nimi. V případě Rails jsou modely primárně využívány pro interakci s příslušnou tabulkou v databázi a pro ukládání pravidel této interakce. Ve většině případů odpovídá jedna tabulka v databázi jednomu modelu aplikaci. Modely obsahují většinu aplikační logiky.

4.2.1.2 Views

Pohledy, neboli views reprezentují uživatelské rozhraní aplikace. V Rails jsou views obvykle HTML soubory s vloženými částmi Ruby kódu, který provádí pouze úkony týkající se prezentace dat. Views mají na starosti poskytování dat webovému prohlížeči nebo jinému nástroji, který zasílá vaší aplikaci požadavky.

4.2.1.3 Controllers

Kontrolory fungují jako zprostředkovatel mezi modely a views. V Rails slouží kontrolory k zpracování požadavků které přichází z webového prohlížeče, získávání dat z modelů a k odesílání těchto dat do views, kde budou zobrazeny.

4.2.2 DRY

DRY (Don't repeat yourself) [6] je princip vývoje softwaru zaměřený na snížení opakování psaní stejného kódu a tím zvyšuje čitelnost a znovupoužitelnost kódu. To znamená, že informace se nacházejí na jednoznačném místě. Pro příklad Ruby on Rails získává definici sloupců pro třídu modelu přímo z databáze.

4.2.3 CoC

CoC (Convention over Configuration) [5] je další princip používaný v Rails pro zlepšení čitelnosti a znovupoužitelnosti kódu. Tento princip znamená, že konvence má přednost před konfigurací a to tak, že Rails předpokládá to, co chcete udělat, místo toho, aby vás nutil specifikovat každou drobnost v konfiguraci.

4.2.4 REST

REST (Representational State Transfer) [12] je architektonický vzor pro webové aplikace. Je založen na HTTP protokolu a hlavní myšlenkou je poskytovat přístup ke zdrojům dat. Všechny zdroje jsou identifikovány přes URI. REST definuje čtyři základní metody pro přístup ke zdrojům. Jsou známy pod označením CRUD³. Tyto metody jsou implementovány pomocí odpovídajících metod HTTP protokolu. Jednotlivé metody rozeberu na příkladech pro zdroj *observations*⁴.

³create, retrieve, update a delete

⁴zdroj aplikace pro práci s hospitacemi

Create je požadavek, který pomocí metody POST vytvoří nový záznam. Příklad dotazu vytvoří novou hospitaci.

```
POST /observations
```

Retrieve je požadavek pro přístup ke zdrojům. Funguje stejným způsobem jako běžný požadavek na stránku pomocí GET metody. V prvním příklad vrátí seznam všech hospitací. Druhý příklad vrátí podrobnosti hospitace s id 1.

```
GET /observations  
GET /observations/1
```

Update je požadavek, pro upravení konkrétního záznamu přes metodu PUT.

```
PUT /observations/1
```

Delete je požadavek, který smaže konkrétní záznam pomocí DELETE metody.

```
DELETE /observations/1
```

4.3 Struktura aplikace

V této sekci popíšu základní strukturu aplikace. Struktura aplikace je vygenerovaná pomocí generátorů v Ruby on Rails a proto nepopíšu celou strukturu, ale pouze části aplikace, které byli nejdůležitější pro vývoj aplikace. Na obrázku [4.1](#) je popsána struktura složek a k nim v popisku jejich obsah.

```

Hospitace/
├── app/
│   ├── assets/ .....obsahuje obrázky, JavaScript, CSS
│   ├── controllers/ .....controllers aplikace
│   ├── helpers/ .....pomocné funkce pro vytváření pohledů
│   ├── inputs/ .....speciální vstupní položky pro tvorbu formulářů
│   ├── mailers/ .....třídy které generují a odesílají emaily
│   ├── models/ .....modely aplikace
│   └── views/ .....šablony aplikace
├── lib/.....rozšíření a moduly pro aplikaci
│   ├── email_templates/ .....modul pro generování emailů
│   ├── tasks/ .....obsahuje rake scripty
│   ├── kosapi/ .....knihovna pro připojení ke KOSapi
│   └── will_paginate/ .....rozšíření pro modul will_paginate
├── public/ .....složka, která je přístupná přes web. Obsahuje statické soubory
├── config/ .....konfigurační soubory a směrování
├── db/ .....schéma databáze a databázové migrace
└── test/ .....testy, testovací data a nástroje pro testování aplikace

```

Obrázek 4.1: Struktura aplikace

Kapitola 5

Realizace

Po dohodě s vedoucím práce byl zvolen iterační vývoj aplikace. Za účelem postupného vyvíjení částí aplikace, tak aby bylo možné testovat aplikaci na letním semestru 2011/2012. V této kapitole popisuji jednotlivá zadání každé iterace. Jak jsem postupoval k vyřešení zadání a výstupy z jednotlivých iterací. Jednotlivé iterace byly prezentovány na konzultacích s vedoucím práce.

5.1 První iterace

5.1.1 Zadání

Tato iterace patřila k nejobsáhlejší z všech iterací. Jedním z důvodů bylo seznámení z novou technologií Ruby on Rails, s kterou jsem neměl zkušenost. První iterace měla za cíl navrhnout a vytvořit základní architekturu aplikace s napojením na KOSapi [10] a k tomu vytvořit funkční část aplikace pro plánování hospitací, tak abych mohl prezentovat funkčnost.

5.1.2 Postup

5.1.2.1 Datová vrstva

Protože aplikace využívá dva datové zdroje KOSapi viz. 4.1.2 a databázi aplikace, bylo potřeba nejdříve vyřešit jak se připojit ke KOSapi. V této části vycházím z prototypu aplikace pro správu hospitací. Kde využívá již naprogramovanou knihovnu z projektu VyVy [16]. Poté bylo potřeba vytvořit modely tak, aby umožnily komunikaci mezi KOSapi a databází aplikace. Při implementování knihovny do aplikace jsem musel vyřešit lokalizaci jazykových konstant v date získaných z API. Vyřešil jsem to rozšířením knihovny o podporu modul `i18n`¹, který je součástí Rails.

¹ lokalizace softwaru pro různé jazyky a jejich místních zvyklostí

5.1.2.2 Autentizace

Pro autentizaci, v této fázi vývoje, používám knihovnu `authlogic`, kterou jsem zprovoznil pomocí návodu [2]. Tuto knihovnu používám jen dočasně pro vývoj aplikace. Ve finální fázi bude nahrazen autentizační službou FELid viz. 4.1.3, kterou lze zprovoznit jen na serveru, kde bude aplikace nasazena.

5.1.2.3 Autorizace

Autentizace v hospitacích je jedna z kritických oblastí, kterou bylo potřeba vyřešit hned na začátku vývoje. V aplikaci potřebuji autentizovat uživatele podle role, tak i podle vztahu k hospitaci. Proto jsem hledal modul, který by dokázal nadefinovat pravidla autentizace. Knihovnu, kterou jsem použil, se jmenuje `CanCan` [4]. Tato knihovna má velmi jednoduchý ale i přesto flexibilní zápis pravidel, která dokáží filtrovat jak podle zdroje tak i podle jednotlivých záznamů, dokáže filtrovat `controllery` tak i zdroje aplikace. Veškerá pravidla jsou definována na jednom místě² aplikace.

Příklad zapsaného pravidla pomocí `CanCan` v modelu `Ability`. Pravidlo slouží pro administrátora hospitací a umožní mu zobrazovat, vytvářet, upravovat a mazat ze zdroje `Observation`. Zároveň zakáže tyto operace pro záznamy, které nevytvořil.

```
def admin
  can :manage, Observation
  cannot :manage, Observation do |ob|
    !(ob.created_by==current_user)
  end
end
```

5.1.2.4 Role

Role pro jednotlivé uživatele uchovávám v modelu `Role`. Kde jednotlivé role uživatele ukládám do bitové masky. Role v bitové masce jsou reprezentovány mocniny čísla 2 díky tomu lze skládat role pomocí bitové operace OR. V tabulce 5.1 je přehled rolí s číslem reprezentovaným bitovou masku role.

Role	Bitová maska
Administrátor hospitací	1
Hospitující	2
Hospitovaný	4
Admin	8

Tabulka 5.1: Reprezentace rolí v bitové masce

²model `Ability`

5.1.2.5 Uživatelské prostředí

Pro vytvoření uživatelského prostředí jsem použil již existující modul Bootstrap [3]. Použil jsem tuto knihovnu abych si usnadnil implementaci uživatelského prostředí. Knihovna obsahuje kompletní CSS tak i Javascriptové moduly. Mezi další výhody patří licence ta je open-source a další výhodou je známé uživatelské prostředí používané v Twitteru.

Pro usnadnění implementace Bootstrap modulu do aplikace jsem využil další dva moduly pro aplikaci. První je modul SimpleForm [15], slouží k vytváření formulářů. Základním cílem je nadefinovat rozvržení všech formulářů na jednom místě. Druhým modulem je WillPaginate [18] pro stránkování dat.

5.1.3 Výstup

Výstupem z první iterace vznikla část aplikace, která uměla plánovat hospitace a uměla získávat data z webové služby KOSapi.

5.2 Druhá iterace

5.2.1 Zadání

Zadáním druhé iterace bylo na-implementovat hodnotící formuláře hospitací. Požadavkem bylo možnost upravovat formuláře bez zásahu do kódu aplikace. Dalším cílem bylo potřeba vyřešit problém s KOSapi, který vznikl při přechodu na nový semestr. Stalo se to že služba neudrží data instancí předmětů z minulých semestrů a tím nebylo možné získat všechny potřebné informace pro hospitace z minulých semestrů.

5.2.2 Postup

5.2.2.1 Datová vrstva

Problém se ztrátou dat byl velmi vážný, proto bylo potřeba tento problém rychle vyřešit abych mohl pokračovat ve vývoji. Musel jsem proto předělat datovou část. Rozšířil jsem databázi o entity z kapitoly 3.3 tak, aby data z KOSapi byli uloženi v databázi. Data je potřeba synchronizovat v databázi s KOSapi. O synchronizaci se stará rake³ script stará se o přidávání a aktualizaci záznamy. Synchronizační script spustím každý den pomocí programu Cron⁴.

Po přidání nových entit bylo potřeba na-implementovat asociace mezi novými entitami. V doménového modelu na obrázku 3.2 je spousta asociací mezi osobou - paralelkou a osobou - instancí předmětu. Protože všechny tyto asociace⁵ mají kardinalitu N:M. Pokud bych použil klasickou dekompozici přes pomocnou tabulku, která rozloží asociace na 1:N a 1:M. Vzniklo by mi 6 pomocných tabulek. Proto jsem rozhodl použít jiný způsob dekompozice,

³rake je sada nástrojů používané pro vývoj a nasazení Rails aplikací

⁴Cron je program, který na pozadí operačního systému spouští naplánované úlohy

⁵asociace garant, přednášející, vyučující, instruktor a zkoušející.

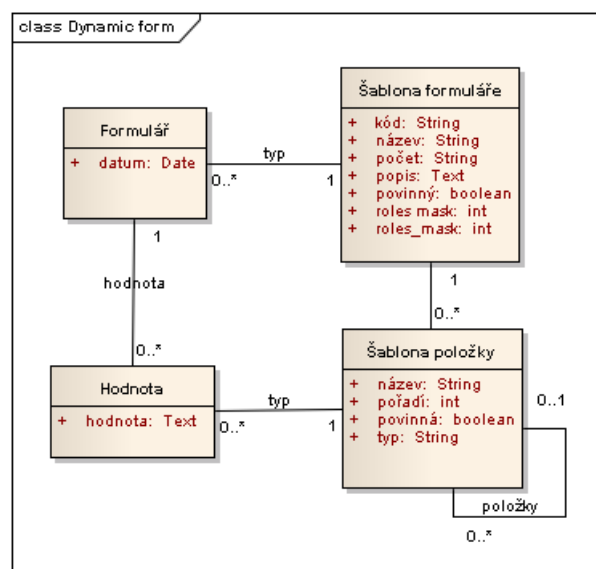
který používá polymorfní asociace [9] v Ruby on Rails. Výhodou této dekompozice je redukce počtu pomocných tabulek. Tento druh asociace zredukoval počet ze 6 na 1 tabulku. V tabulce 5.2 jsou popsány jednotlivé atributy s příklady modelu `PeopleRelated`.

Atribut	Příklad	Popis
<code>related_id</code>	1	cizí klíč záznamu, který je v asociaci s osobou
<code>related_type</code>	Parallel	jméno modelu ke kterému patří cizí klíč
<code>relation</code>	teachers	název asociace
<code>people_id</code>	100	cizí klíč pro osobu

Tabulka 5.2: Popis atributů modelu `PeopleRelated`

5.2.2.2 Hodnotící formuláře

Z důvodu možné změny hodnotících formulářů v budoucnosti. Jsem vytvořil návrh, který umožní vytvářet nové typy formulářů, nebo upravovat již existující. Formuláře se definují šablonou, která definuje základní vlastnosti jako jsou minimální a maximální počet vyplnění, název a kdo může formulář vyplnit. Samotný obsah šablony formuláře se skládá z položek. Položky formuláře mohou být hodnotící tabulka, nadpis, hodnocení, text. Na obrázku 5.1 je doménový model dynamických formulářů.



Obrázek 5.1: Doménový model dynamických formulářů

Hodnotící formuláře se generují podle šablony formuláře získané z databáze. Šablona je reprezentovaná modelem `FormTemplate`. Tato šablona obsahuje veškeré informace potřebné pro definování chování formuláře. Mezi základní vlastnosti, které definuje šablona, patří název, popis a kód. Kód šablony je velmi důležitá vlastnost, definuje skupinu kam patří

formulář. Skupiny formulářů jsou rozděleny podle druhu formuláře a ty popisují v sekci hodnocení výuky 2.3.2.5. Formuláře jsou rozdělené do skupin protože se časem mohou upravovat a nemůžu si dovolit upravit již vytvořenou šablonu. Mohlo by nastat ztráta dat ve vyplněných formulářích, proto musím se musí vždy vytvořit nová šablona.

Šablona formuláře definuje také postupné hodnocení. Pro otevření dalšího formuláře se musí splnit několik podmínek:

Povinný formulář musí být vyplněn a také musí splnit podmínku pro minimální počet. V jiném případě není potřeba vyplnit žádný formulář.

Minimální počet lze definovat buď přesným počtem, nebo dynamicky podle asociace. Kde minimální počet mi definuje počet instancí v asociaci. Počet vytvořených formulářů se počítá pouze z jednoho hodnocení hospitace.

Hospitující i hospitovaný může vyplnit maximálně jeden formulář.

U dynamických formulářů jsem musel také řešit, jejich chování pro různé role v systému. Proto jsem přidal do šablony dvě bytové masky jednu pro zobrazování a druhou pro vyplňování formulářů. Používám zde princip pro identifikaci rolí viz. 5.1.2.4.

Protože formulář Práva pro vytváření i zobrazování formulářů jsem použil stejný způsob identifikace rolí, který využívám pro role osob. Jsou obsaženy v šabloně Používám dvě masky v jedna identifikuje role pro tvorbu formulářů a druhá definuje role které mohou zobrazit formulář.

Typ	Návratová hodnota	Popis
label		textový popis
integer	číslo	vstupní element pro čísla
text	text	formulář pro psaní textů
text/file	text	formulář pro psaní textů s možností nahrání souboru s naskenovaným formulářem
ranking_table		tabulka pro hodnocení, může obsahovat několik elementů typu ranking
column_table		tabulka do které se vkládají jiné elementy po sloupcích
ranking	[A,B,C,D,E,F]	vstupní element pro zadávání známkování od A do F
ranking_scale		tabulka s hodnotící stupnicí
note	text	vstupní element pro napsání textové poznámky

Tabulka 5.3: Seznam typů elementů

Pro vygenerování obsahu formuláře používám položky formuláře reprezentované modelem **EntryTemplate**. Položky formuláře definuje co se má vykreslit a kde. Umístění jednotlivých položek v dokumentu je definované ve stromové struktuře. Kde kořenem stromu je šablona formuláře a ta se postupně větví přes asociaci podpoložka. Co se má vykreslit je definované atributem typu elementu. Seznam podporovaných typů elementů a jejich vlastnosti jsou

vypsány v tabulce 5.3. Samotné generování obsahu není složitý proces. Celý formulář se sestaví hierarchicky i s hodnotami z modelu **Entry**.

Nestačí nám pouze vygenerovat formulář ale potřebujeme i uložit jeho hodnoty do databáze o to se starají modely **Form** a **Entry**. Model **Form** složí pro identifikaci konkrétního vyplněného formuláře. Jednotlivé hodnoty z vyplněného formuláře jsou uloženy v **Entry**.

5.2.3 Výstup

Výstupem této iterace byla aplikace, která již využívala pouze svoji databázi pro zdroj dat z KOSu a dynamických formulářů s nadefinovanými šablonami hodnotících formulářů.

5.3 Třetí iterace

5.3.1 Zadání

Zadáním třetí iterace bylo připravit server pro službu FELid. Dalším požadavkem bylo zprovoznění automatické zálohování databáze.

5.3.2 Postup

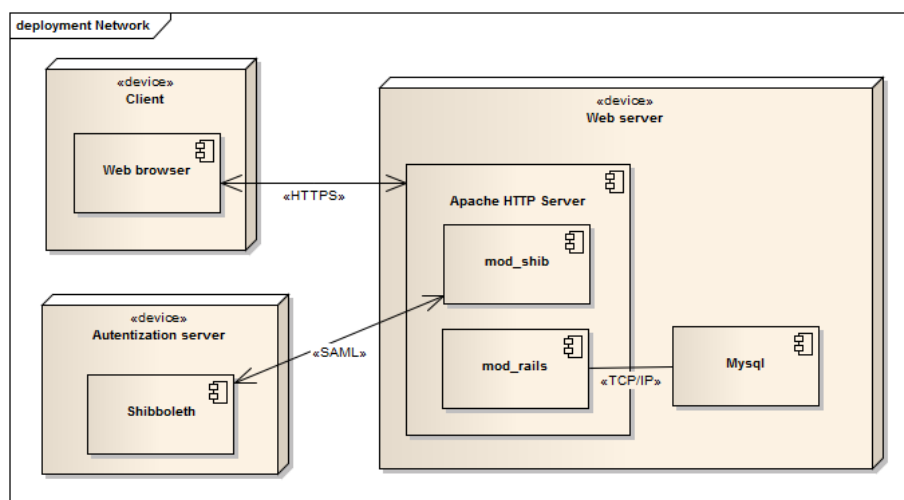
5.3.2.1 Nasazení aplikace

Na diagramu nasazení 5.2 jsou znázorněny komponenty potřebných pro rozjetí aplikace. Protože na serveru byly nainstalovány pouze webový server Apache HTTP Server a databázový systém Mysql, bylo potřeba nainstalovat platformu Ruby a framework Ruby on Rail. K instalaci Ruby jsem se rozhodl použít software RVM. Instalaci jsem provedl podle návodu pro více uživatelskou instalaci [1]. Ten umožní nainstalovat různé platformy Ruby na jednom počítači. Mezi další přínosem je spravování gemsets⁶, díky nim lze zprovoznit několik aplikací zároveň na jednom počítači bez problému. Do nainstalovaného webového serveru bylo potřeba ještě přidat zásuvný modul, který umožní nasadit Rails aplikace. Modul se jmenuje Passenger (mod_rails) a instaloval jsem ho podle návodu pro instalaci do Apache [26].

5.3.2.2 FELid

Pro zprovoznění služby FELid, bylo potřeba splnit několik požadavků. Tyto požadavky jsou umístěny na oficiálních stránkách služby [8]. Jeden z požadavků je zprovoznění programu, který zajistí komunikaci se službou FELid. Tento program se jmenuje Shibboleth a obsahuje zásuvný modul do webového serveru mod_shib. Konfiguraci programu jsem postupoval podle návodu, který se nachází na wiki stránkách FELid [7]. Po splnění všech požadavků stačilo jen zažádat o zaregistrování aplikace.

⁶balíky knihoven mezi kterými lze přepínat pro různé aplikace



Obrázek 5.2: Diagram nasazení

5.3.2.3 Záloha databáze

Pro automatické zálohování databáze jsem použil existující ruby aplikaci pro zálohování. Tato aplikace každý den vytvoří zálohu databáze, kterou uloží na disk. Na disku uchovává 300 záloh databáze, při překročení počtu záloh se starší zálohy nahrazují novými. Výhodou tohoto řešení je podpora různých databázových systémů a možnost ukládat zálohy na externích uložisti⁷ pro uchování záloh. Záloha se spouští pravidelně každý den brzo ráno pomocí příkazu v konzoli:

```
backup perform --trigger backup
```

5.3.3 Výstup

V této iteraci jsem se mi podařilo zprovoznit aplikaci na serveru a otestovat její funkčnost. Také jsem pracoval na programu, který jsem upravoval podle získaných připomínek od vedoucího práce.

5.4 Čtvrtá iterace

5.4.1 Zadání

Tato iterace je poslední a tím většinu zadání tvořily připomínky od vedoucího práce. Hlavními úkoly v této iteraci je vytvořit modul, který bude generovat emailové zprávy po vyplnění hodnotícího formuláře a integrovat autentizaci přes FELid.

⁷ pro příklad Amazon Simple Storage Service (S3)

5.4.2 Postup

5.4.2.1 Autentizace

V této fázi vývoje byla aplikace zaregistrovaná ve FELid. Před integrování autentizace přes FELid, je potřeba otestovat aplikaci. Nejdůležitější je otestovat správnou funkčnost práv pro různé role. Pro zprovoznění autentizace přes FELid může jednoduše testovat aplikaci pod různými uživateli. Od zprovoznění nebudu mít totiž kontrolu nad autentizací, tu zajišťuje služba FELid.

Integrace do aplikace je pak velmi jednoduchá. Stačilo jen odstranit dočasný způsob autentizace, který jsem implementoval v první iteraci. Systému FELid poskytuje informace o přihlášeném uživateli prostřednictvím atributů v `request.env`. Pro identifikaci přihlášeného uživatele používám atribut `felid-uid` ve kterém se nachází uživatelské jméno. Autentizaci v aplikaci se stará jen jedna funkce `current_user_session`.

```
def current_user_session
  return @user if defined?(@user)
  if not request.local? and not request.env["felid-uid"].nil?
    and @user.nil?
  then
    @user = People.find_by_username request.env["felid-uid"]
  end
  return @user
end
```

5.4.2.2 Emaily

Jeden z požadavků bylo odesílání emailových zpráv po vyplnění hodnotícího formuláře. Součástí zadání je generování emailů ze šablon. Ty půjde upravovat. Šablona se skládá z předmětu, textu zprávy a název akce. Akce v systému mohou být různých druhů, podle zadání stačilo pouze implementovat akce pro vytvoření formulářů. Při vykonání akce se vygeneruje emailová správa ze šablony emailu.

5.4.2.3 Generátor obsahu emailových zpráv

Protože je potřeba generovat do emailu proměnná data v hospitaci. Tak jsem vytvořil modul `EmailTemplates` pro generování emailů. Úkolem modulu je nahradit značky v textu za skutečná data. Pro vložení generovaných dat se používá zápis `%značka%` do textu zprávy.

Příklad vygenerování jednoduchého emailu se značkou pro kód předmětu.

```
text %course_code% => text A7B36ASS
```

Při vývoji modulu jsem si dal za cíl vytvořit takový modul který by umožnil jednoduše nastavovat podporované značky. Největším zádrhelem v generování obsahu zprávy je v získání dat z hospitací. Vstupem do generátoru je pouze instance šablony a hodnocení hospitace. Ostatní data je potřeba získat z různých asociací.

Modul obsahuje dva pod moduly. Prvním částí je modul `ModelHelper`, který slouží pro rozšíření libovolného modelu v aplikaci. Rozšíření umožní pomocí metody `attrs_tagged(*args)` povolit atributy, které bude možné použít pro generování emailů.

Příklad jednoduchého nastavení modelu `Person`, které umožní generátoru emailů vkládat jméno, emailovou adresu a uživatelské jméno.

```
class Person < ActiveRecord::Base
  include EmailTemplates::Tagged::ModelHelpers
  attrs_tagged :name, :email, :username
end
```

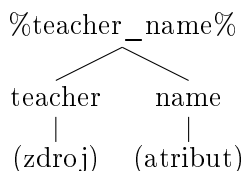
Druhou částí je modul `EmailBuilder`. Tento modul pomáhá si vytvořit vlastní generátor pomocí jednoduchého nastavování. Modul poskytuje metodu `source(model, name, *path)` pro nastavení jednotlivých zdrojů dat. Metoda využívá k nastavení generátoru nastavení z modelu, který byl rozšířen modulem `ModelHelper`, název zdroje a cestu k instancím modelu s daty. Cesta je definovaná posloupností asociací mezi modely.

Příklad implementace jednoduchého generátoru `EmailBuilder`, který dokáže vkládat do emailů informace o hospitovaném učiteli.

```
class EmailBuilder
  include Tagged::EmailBuilder
  source Person, :teacher, :evaluation, :teacher
end
```

Tento generátor bude podporovat tři značky `%teacher_name%`, `%teacher_email%` a `%teacher_username%`. Tímto způsobem lze jednoduše nadefinovat velké množství značek.

Struktura značky 5.3 se skládají ze dvou základních částí ze jména zdroje a jména atributu, ze kterého generátor získá data pro nahrazení značky.



Obrázek 5.3: Struktura značky

Výhodou tohoto modulu je krátký zápis a jednoduché nastavování generátoru. V aplikaci jsem celkem nadefinoval pro 9 zdrojů 32 značek.

5.4.2.4 Odesílání emailů

Podle zadání stačilo implementovat odesílání zpráv po akci vytvoření hodnotícího formuláře. Podle druhu hodnotícího formuláře se vygeneruje zpráva a odešle se administrátorovy hospitace, hospitujícím, hospitovaným garantovi předmětu a vedoucímu katedry.

Z časových a technických problémů se mi nepodařilo zprovoznit emailový server pro odesílání emailů, proto dočasně používám k odesílání emailový účet hospitace-FEL-CVUT@gmail.com na Gmail.

5.4.3 Výstup

Výstupem z této iterace je finální aplikace, které splnila všechny hlavní požadavky.

Kapitola 6

Testování

Cílem testování bylo otestovat reálnou funkčnost aplikace a odhalit vzniklé chyby při vývoji aplikace. Aplikaci jsem testoval dvěma způsoby manuálně, kde jsem musel ručně procházet aplikaci a zkoušet její funkčnost, a Funkčnost aplikace jsem testoval manuálně a automaticky pomocí testovacích nástrojů v Ruby on Rails. Aplikaci jsem testoval v průběhu celého vývoje.

6.1 Automatické testování

Automatické testy je důležité používat na místech, kde je velké riziko vzniku chyb. Nejčastější vznikají chyby při přidávání, nebo úpravě funkcionality. Pro automatické testování jsem použil testovací nástroje v Ruby on Rails. V Ruby on Rails se testy dělí do tří kategorií.

6.1.1 Testovací data

Pro automatické testování je potřeba připravit testovací data. Testovací data jsou obsažena ve struktuře programu pod tests/fixtures. Zde se pak nacházejí Soubory s testovacími daty pro jednotlivé modely ve formátu YAML.

6.1.2 Unit testing

Unit testy slouží k testování samostatných částí programů. V Rails se Unit testy používají hlavně pro testování funkčnosti modelů. Testuje se hlavně validace vstupů a perzistence dat.

6.1.3 Functional tests

Tyto testy testují rozné činnosti v jednotlivých controllerech aplikace. Controllery zpracovávají příchozí webové requesty a nakonec odpoví vyrendrovaným pohledem. Tento typ testů testuje:

- Byl webový požadavek úspěšný?
- Byli jsme přesměrováni na správnou stránku?

- Byli jsme úspěšně přihlášení?
- Byl objekt vložen do správné šablony?
- Byla zobrazena správná hláška uživateli?

6.1.4 Integration testing

Integrační testy se používají k testování interakce mezi libovolným počtem kontrolorů. Obvykle se používají k testování důležitých pracovních toků v rámci aplikace.

6.2 Manuální testování

Manuálně jsem testoval části aplikace u kterých se špatně vytvářejí automatické testy, nebo byl potřeba lidský úsudek¹. Primárně jsem testoval reálnou funkčnost aplikace na konci každé iterace. Procházel jsem aplikaci podle scénářů užití a využíval jsem reálná data z probíhajících hospitací v letním semestru 2011/2012.

¹testování uživatelského prostředí

Kapitola 7

Závěr

Výstupem mé práce je aplikace pro evidenci hospitací. Aplikaci jsem nasadil na server <http://kvalitavyuky.felk.cvut.cz>, tak že ji lze použít v reálném provozu na ČVUT FEL. Samotná aplikace byla implementovaná na platformě Ruby on Rails, což je jeden z nejmodernějších frameworků pro vývoj webových aplikací. Bylo potřeba v aplikaci používat dvě fakultní aplikace. Pro autentizaci používá aplikaci FELid je globální autentizační systém pro webové aplikace na fakultě FEL. Druhou aplikací je KOSapi, ta poskytuje aplikační rozhraní k přístupu dat v KOSu.

Pro vývoj aplikace byl použit iterační vývoj, kdy při každé iteraci byly zpracovány požadavky zadavatele a zdokumentovaný. Snažil jsem se vytvořit funkční a uživatelsky přívětivou aplikaci. Funkčnost aplikace jsem demonstroval simulací probíhajících hospitací pro studijní program STM v LS 2011/2012. Bohužel se nepodařilo aplikaci nasadit do reálného provozu, aby se vyzkoušela v reálném provozu.

Tato práce mi přinesla spoustu zkušenosti jak s novou technologií Ruby on Rail, kterou jsem do té doby nepoužil, tak i použití externích aplikací. Při vývoji jsem také vytvořil dvě velké části aplikace. Jsou to dynamické formuláře pro hodnocení a knihovna pro generování emailových zpráv.

7.1 Možnosti v pokračování práce

Možností pokračování v práci je několik. Nejvíce je potřeba nainstalovat a nastavit na serveru emailový server pro odesílání informačních emailových zpráv.

Lze také pokračovat ve vývoji dynamických formulářů. Do aplikace jsem totiž neimplementoval uživatelské prostředí pro nastavování formulářů. Dále by bylo dobré předělat funkční část do knihovny. Jako dalším krokem pro vylepšení aplikace bych to vyděl na provedení refaktoring celé aplikace, protože při vývoji jsem se postupně učil pracovat v Ruby on Rails.

Literatura

- [1] *Installing RVM* [online]. [cit. 2012-04-07]. Dostupné z: <<https://rvm.io/rvm/install/>>.
- [2] *Authlogic: documentation* [online]. [cit. 2012-04-07]. Dostupné z: <<http://rdoc.info/github/binarylogic/authlogic>>.
- [3] *Bootstrap, from Twitter* [online]. 2012-04-07. Dostupné z: <<http://twitter.github.com/bootstrap/>>.
- [4] *CanCan: Getting Started* [online]. 2012. Dostupné z: <<https://github.com/ryanb/cancan>>.
- [5] *Convention over configuration*. In: *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 20.9.2007, last modified on 10.12.2011. [cit. 2012-03-05]. Dostupné z: <http://en.wikipedia.org/wiki/Convention_over_configuration>.
- [6] *Don't repeat yourself*. In: *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 1.12.2005, last modified on 9.2.2012. [cit. 2012-03-05]. Dostupné z: <http://en.wikipedia.org/wiki/Don't_repeat_yourself>.
- [7] *O systému FELid* [online]. [cit. 2012-02-15]. Dostupné z: <<http://wiki.feld.cvut.cz/net/felid/about>>.
- [8] *Požadavky FELid* [online]. 2012-02-15. Dostupné z: <<http://wiki.feld.cvut.cz/net/admin/aai/provoz/pozadavky>>.
- [9] *A Guide to Active Record Associations* [online]. [cit. 2012-04-07]. Dostupné z: <http://guides.rubyonrails.org/association_basics.html>.
- [10] *Representational state transfer*. In: *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 17.8.2004, last modified on 3.4.2011. [cit. 2012-03-05]. Dostupné z: <http://en.wikipedia.org/wiki/Representational_state_transfer>.
- [11] *Model-view-controller*. In: *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 5.11.2003, last modified on 5.4.2012. [cit. 2012-04-06]. Dostupné z: <<http://cs.wikipedia.org/wiki/Model-view-controller>>.

- [12] *Representational state transfer*. In: *Wikipedia: the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 17.8.2004, last modified on 3.4.2011. [cit. 2012-03-05]. Dostupné z: <http://en.wikipedia.org/wiki/Representational_state_transfer>.
- [13] *Ruby* [online]. Dostupné z: <<http://www.ruby-lang.org/en/>>.
- [14] *Ruby on Rails* [online]. [cit. 2011-12-15]. Dostupné z: <<http://rubyonrails.org/>>.
- [15] *SimpleForm: Rails forms made easy* [online]. [cit. 2012-04-07]. Dostupné z: <https://github.com/plataformatec/simple_form>.
- [16] *Aplikace Vykazování výuky* [online]. 2012. Dostupné z: <<https://vyvy.felk.cvut.cz/>>.
- [17] *Stránky projektu VyVy* [online]. 2012. Dostupné z: <<http://code.google.com/p/vykazovani-vyuky-cvut/>>.
- [18] *Will_paginate: documentation* [online]. 2012-04-07. Dostupné z: <https://github.com/mislav/will_paginate/wiki>.
- [19] ORACLE CORPORATION. *MySQL 5.5* [software]. Dostupné z: <<http://dev.mysql.com/downloads/mysql/>>.
- [20] APACHE SOFTWARE FOUNDATION. *Apache HTTP server 2.4* [software]. [přístup 2012-04-07]. Dostupné z: <<http://httpd.apache.org/download.cgi>>.
- [21] D. RICHARD HIPPE. *SQLite* [software]. Dostupné z: <<http://www.sqlite.org/download.html>>.
- [22] HLAVÁČ, V. – KOSTLIVÁ, J. *Postupy pro kontrolu kvality výuky: nejen pro studijní program STM*. verze 04. 19. listopadu 2010. Dostupné z: <https://wiki.feld.cvut.cz/_media/rada_stm/2011-04-05kontrolakvalityvyuky_verze_5.pdf>.
- [23] INTERNET2. *Shibboleth SP 2.4* [software]. Dostupné z: <<http://shibboleth.internet2.edu/downloads.html>>.
- [24] KOMÁREK, M. *Kontrola kvality výuky* [online]. [cit. 2012-01-17]. Dostupné z: <https://wiki.feld.cvut.cz/rada_stm/kontrolavyukyverejne>.
- [25] KREŽELOK, D. *Návrh a implementace systému pro správu hospitací*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta elektrotechnická, Praha, 2010.
- [26] PHUSION. *Passenger* [software]. Dostupné z: <<http://www.modrails.com/install.html>>.

Příloha A

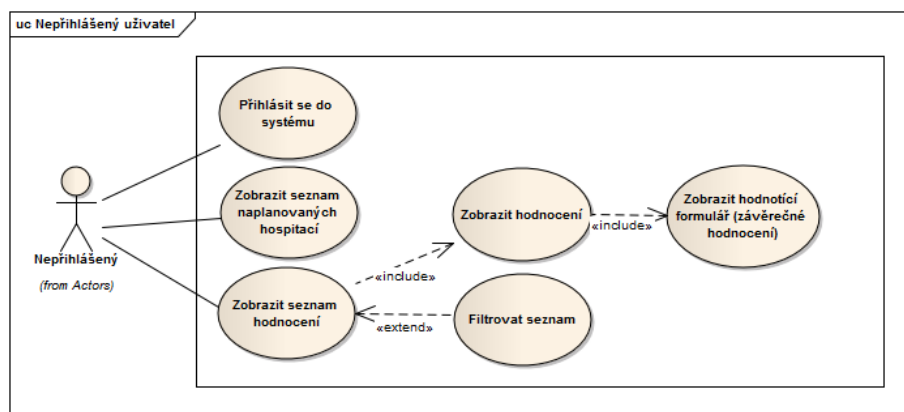
Seznam použitých zkratek

API	Application Programming Interface
CoC	Convention over Configuration
CRUD	Create Retrieve Update Delete
CSS	Cascading Style Sheets
ČVUT	České vysoké učení technické v Praze
DRY	Don't repeat yourself
FEL	Fakulta elektrotechnická
i18n	Internationalization and localization
KOS	Komponenta studium
LS	Letní semestr
MVC	Model-view-controller
HTML	HyperText Markup Language
REST	Representational State Transfer
RVM	Ruby Version Manager
SAML	Security Assertion Markup Language
STM	Softwarové technologie a management
URI	Uniform Resource Identifier
VyVy	Vykazování Výuky
WWW	WorldWideWeb
YAML	Ain't Markup Language

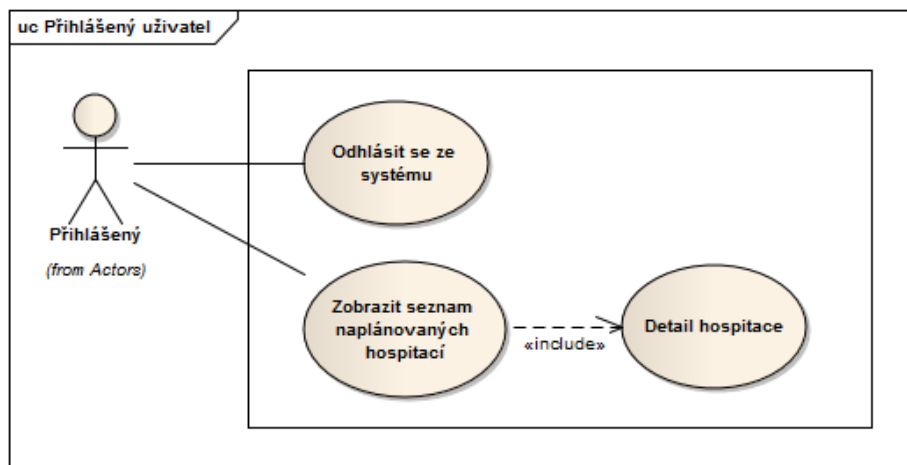
Příloha B

UML diagramy

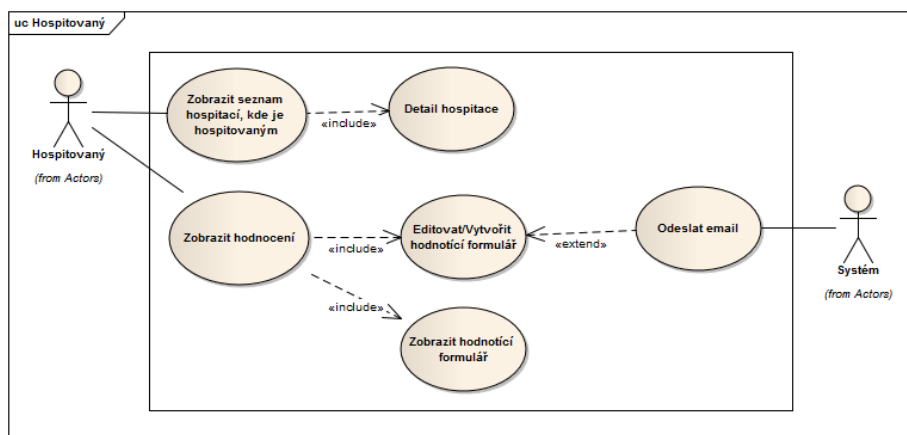
B.1 Use cases



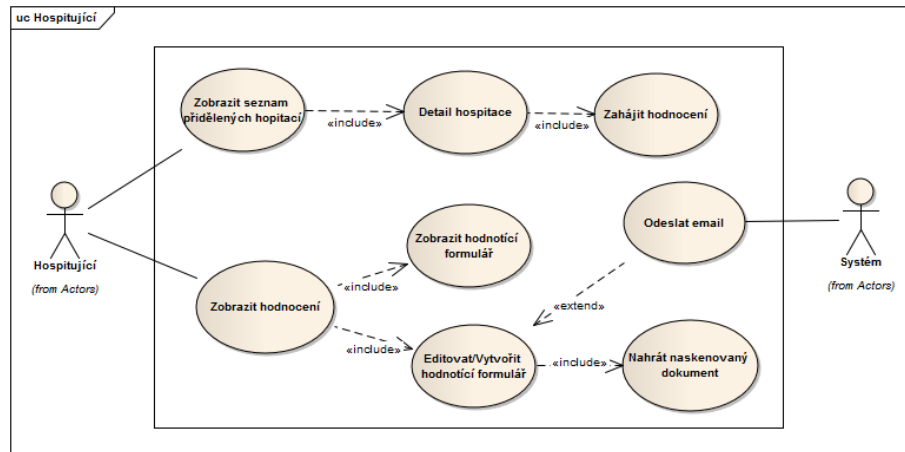
Obrázek B.1: Use case - nepřihlášený uživatel



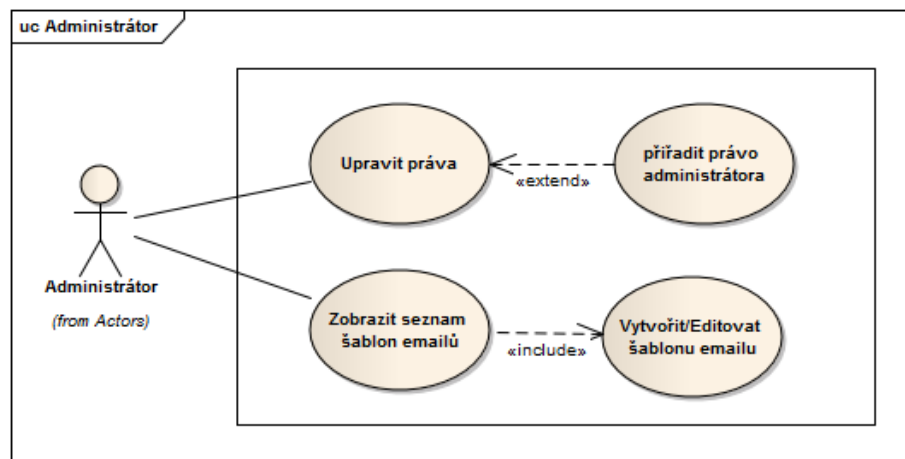
Obrázek B.2: Use case - přihlášený uživatel



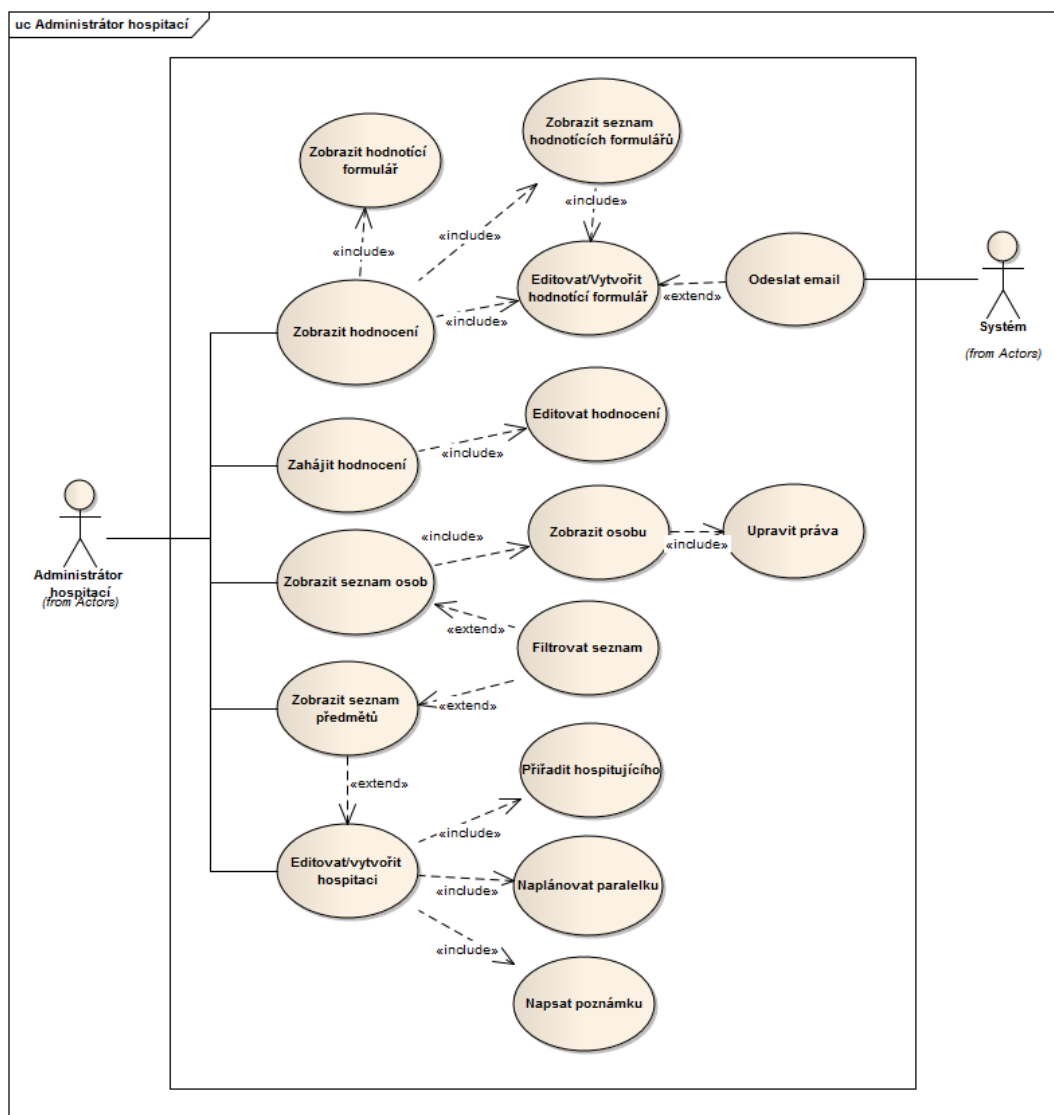
Obrázek B.3: Use case - hospitovaný



Obrázek B.4: Use case - hospitující



Obrázek B.5: Use case - administrátor



Obrázek B.6: Use case - administrátor hospitací