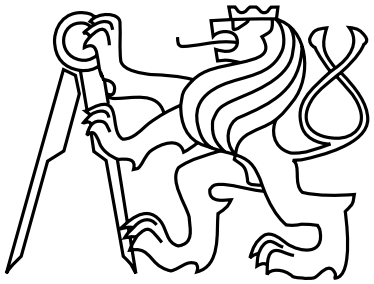
Systém pro evidenci hospitací v Ruby on Rails

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 zapracovávejte požadavky zadavatele. Celý vývoj řádně dokumentujte a výsledky své práce otestujte. Funkčnost systému demonstrujte 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

18. 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 CTU in Prague 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 používaná na ČVUT v Praze 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 vytvořená 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 hospitací v letním semestru 2011/2012.

Obsah

Seznam obrázků

Seznam tabulek

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 výukách[1]. 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.

Cílem mé práce je navrhnout a vytvořit informační systém pro správu hospitací na FEL ČVUT, 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 falkultní aplikace FELid a KOSapi.

[1] zkráceně 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. Zajišťuje jí garantem studijního oboru přiřazený 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 naplánování hospitace musí administrátor ručně obeslat emailem informaci o naplánované 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. Informace proto čerpám ze 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 dokumentace a z rozpracované webové aplikace. Dokumentace k prototypu obsahuje analýzu a návrh aplikace, což mi pomohlo při vývoji aplikace a také pochopení problematiky hospitací. Práce se samotnou aplikací byla již obtížnější, protože verze, kterou jsem obdržel byla velmi rozpracovaná. Nepodařilo se mi jí zprovoznit tak, aby byla plně funkční. Po hlubším zkoumání a upravování aplikace jsem zjistil, že aplikace umí částečně plánovat hospitace. [?co neumí?] Dále byli vytvořeny html šablony pro hodnotící formuláře a knihovna pro komunikaci s aplikací KOSapi[1].

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 se 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 popisuji 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

Ke každému studijnímu programu je přidělena určitá osoba, kterou vybírá garant studijního oboru. Tato osoba se stará o hospitace pro studijní obor ke kterému je přiřazena. Hlavními úkoly administrátora jsou plánování a řízení hospitací.

2.3.2.2 Plánování hospitací

Pro naplánování hospitace musí administrátor hospitací nadefinovat předmět, paralelku a datum hospitace. Hospitace může probíhat jak na přednáškách, tak i na cvičeních. Do plánování patří i přiřazení hospitujících[2], kteří provedou kontrolní návštěvu ve výuce.

2.3.2.3 Typy hospitací

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

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.

[1] viz. 4.1.2

[2] typicky je to dvojce pedagogů jeden zkušený a druhý začínající

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

Předem neohlášené tento typ hospitace se používá jen zřídka a to u problémových předmětů, nebo při vážných stížností 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 ve výuce, o které sepíší 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 sepíše po provedení hospitace 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 a nejdůležitější dokument 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 aplikace. 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. Požadavky na systém jsem převzal z prototypu aplikace [25] a rozšířil jsem je o rozšiřující specifikace zadavatele. Tyto požadavky mi definují návrh aplikace a 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.

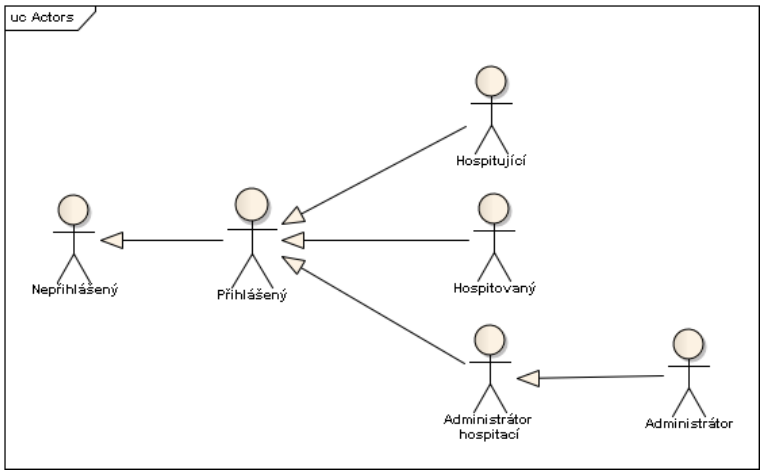
1. Systém umožní průběžně plánovat hospitace.
2. Systém umožní hospitujícímu i hospitovanému prohlížet hospitace.
3. Systém umožní vystavit závěrečné hodnocení na veřejné části aplikace.
4. Systém umožní hospitovanému sepsat stanoviska k názorům hospitujícího.

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 informačním 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 z emailových šablon.
14. Systém bude automaticky zálohovat databázi.

3.2 Uživatelské role

V systému je celkem 7 uživatelských rolí. Definoval jsem čtyři základní uživatelské role, které jsou základem systému: nepřihlášený uživatel, přihlášený uživatel, administrátor hospitací a administrátor. Další dvě role hospitovaný a hospitující se přidělují v rámci konkrétních hospitací. Na obrázku 3.1 jsou vidět jednotlivý aktéři a jejich zobecnění. V další části této sekce rozeberu jednotlivé role a k nim případy užití. Případy užití jsou obsaženy v příloze B.1 tohoto dokumentu.

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.



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. Případy užití pro nepřihlášeného uživatele jsou na obrázku B.1.

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. Případy užití pro přihlášeného uživatele jsou na obrázku B.2.

3.2.3 Hospitovaný

Hospitovaný je role pro přihlášeného uživatele v systému. Je přidělena automaticky pro každého vyučujícího, který vyučuje předmět, na němž byla naplánovaná hospitace a tato hospitace proběhla. Případy užití pro hospitovaného jsou na obrázku B.3.

3.2.4 Hospitující

Hospitující je role pro přihlášeného uživatele v systému. Přiděluje se automaticky osobám, které mají za úkol vykonat hospitaci předmětu. Případy užití pro hospitujícího jsou na obrázku B.4.

3.2.5 Administrátor hospitací

Hlavním úkolem této role je plánovat hospitace na předměty a posléze je spravovat. Případy užití pro administrátora jsou na obrázku B.6.

3.2.6 Administrátor

Administrátor je super uživatel, který má nejvyšší pravomoc v systému. Má přístup ke všem zdrojům aplikace a může aplikaci spravovat. Případy užití pro administrátora jsou na obrázku B.5.

3.3 Doménový model

Doménový model na obrázku 3.2 reprezentuje entity v systému a jejich vzájemné vztahy. Popis jednotlivých domény jsem pro přehlednost rozdělil podle zdroje na dvě základní skupiny. V první skupině jsou domény, které jsem převzal z RESTful rozhraní KOSapi[1] a druhou skupinou jsou domény specifické pro moji aplikaci.

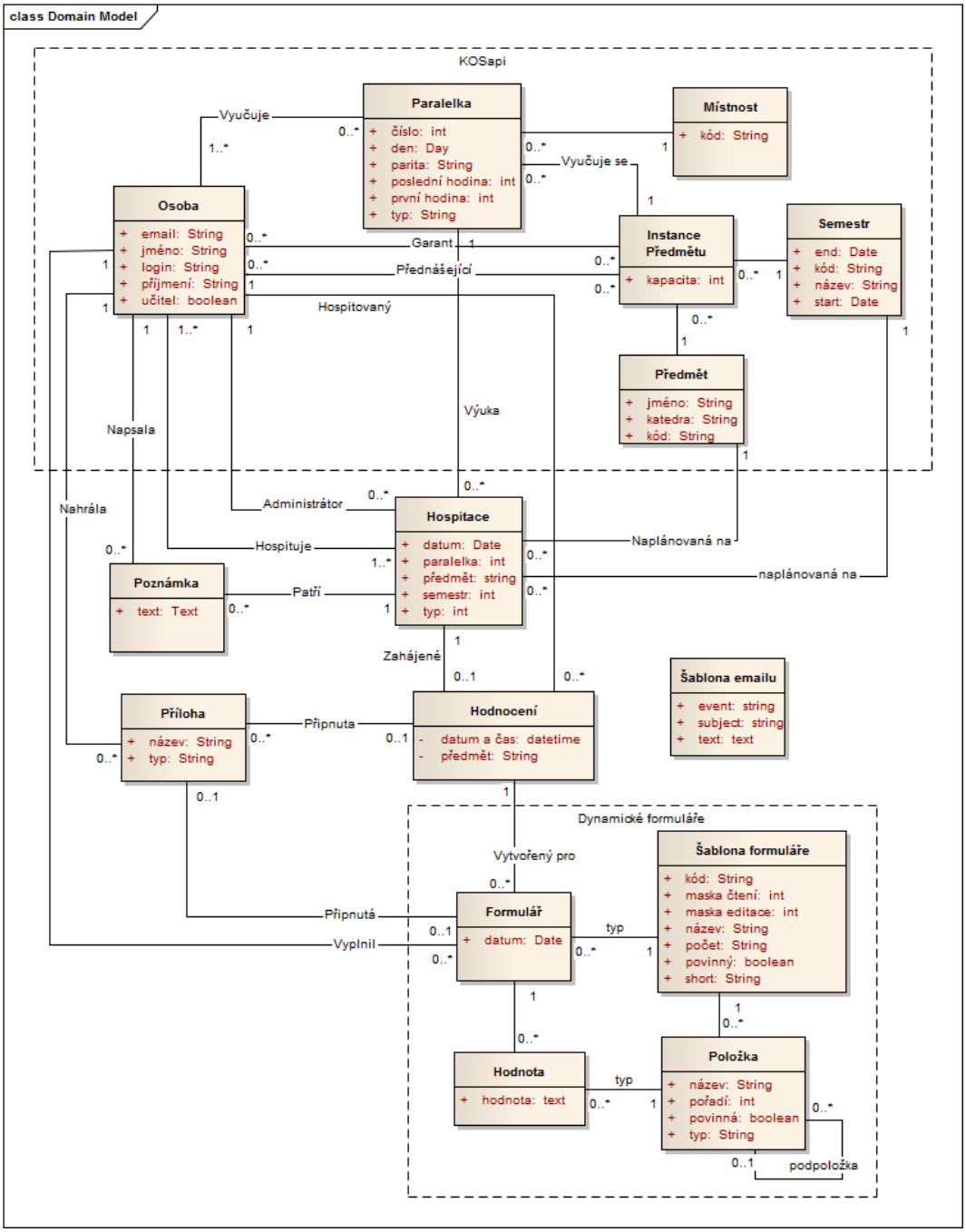
3.3.1 Domény v KOSapi

- 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 vypsané v konkrétním semestru.
- Paralelka - je vypsaná 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 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.

[1] viz. 4.1.2



Obrázek 3.2: Doménový model

3.4 Průběh hospitace

Cílem této části analýzy je popsat průběh hospitace od jejího vytvoření po ukončení. Průběh hospitace je graficky zobrazený na diagramu 3.3.

3.4.1 Vytvoření

Životní cyklus hospitace začíná jejím vytvořením. V této fázi účinkuje pouze aktér administrátor hospitací. Při vytváření hospitace je potřeba definovat: předmět, semestr a typ hospitace[2]. Po úspěšném vytvoření hospitace následuje fáze plánování.

3.4.2 Plánování

Při plánování je také hlavním aktérem administrátor hospitace. V této části životního cyklu administrátor určí paralelku předmětu a datum uskutečnění hospitace. V této fázi musí administrátor hospitací přidělit hospitujícího.

3.4.3 Hodnocení

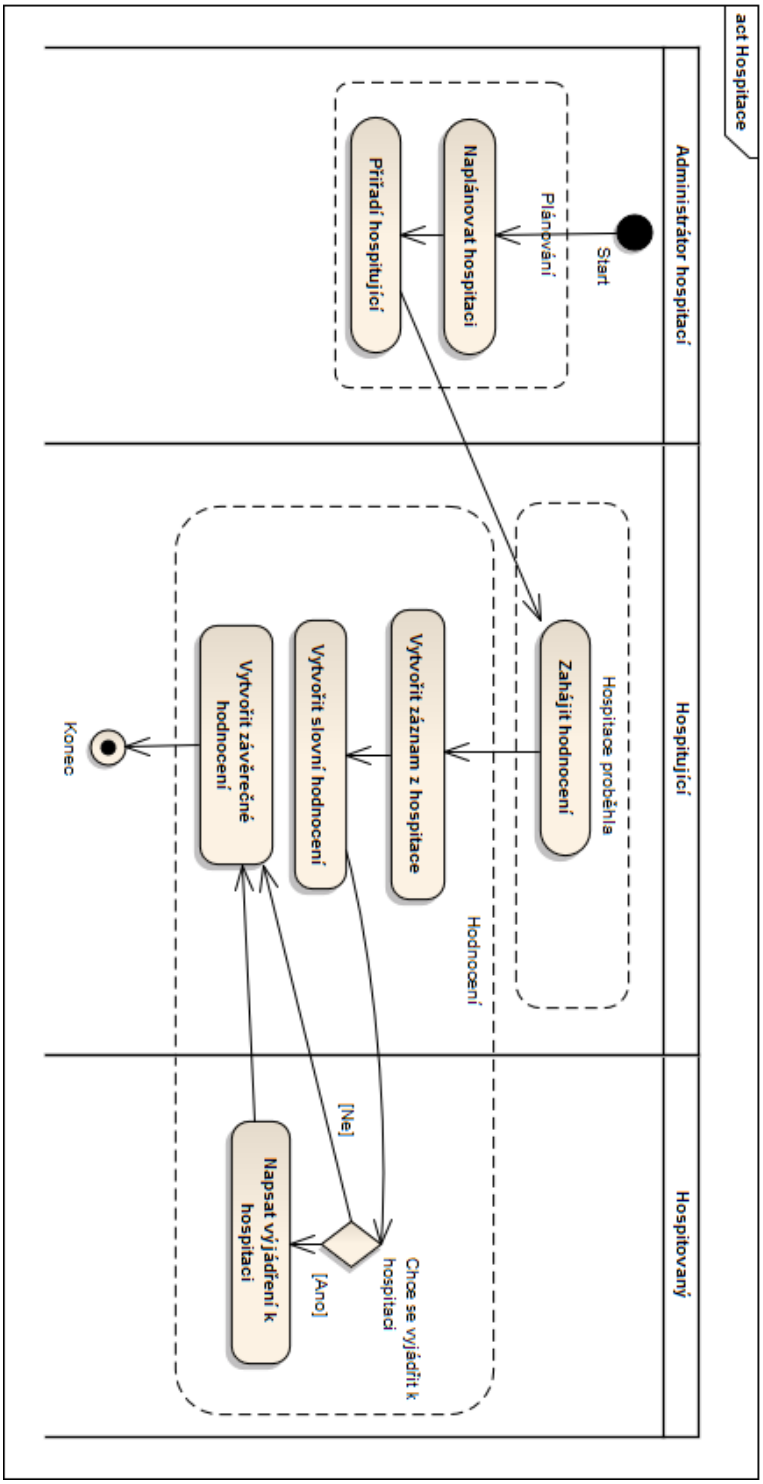
Po vykonání kontrolní návštěvy začíná nová fáze hodnocení vyučování. V této fázi již neúčinkuje administrátor hospitace, ale přicházejí na scénu aktéři hospitovaný a hospitující. Hodnocení probíhá postupným vyplňováním hodnotících formulářů.

1. Hospitující musí vyplnit, nebo nahrát naskenovaný formulář Hodnocení výuky při hospitaci. Tento formulář slouží k dokumentaci průběhu hospitace.
2. Jeden z hospitujících sepíše Slovní hodnocení hospitační návštěvy.
3. Pokud hospitovaný se chce vyjádřit k hospitaci, může vyplnit formulář Stanovisko hodnoceného k názorům hospitujícího.
4. Jeden z hospitujících sepíše formulář Závěrečné shrnutí.

3.4.4 Ukončená

Po vyplnění posledního formuláře Závěrečné shrnutí je hospitace ve stavu ukončená a tím končí i její životní cyklus.

[2] definuje způsob zviditelnění, pro ostatní aktéry v aplikaci



Obrázek 3.3: Průběh 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 v zadání práce. Jedná se o moderní framework určený pro vývoj webových aplikací napojených na relační databázi. Framework je napsán ve skriptovacím interpretovaným programovacím jazyku Ruby [13]. Rails používá návrhový vzor Model-view-controller viz 4.2.1. Mezi základní myšlenky 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 Balíckovací systém RubyGems

Protože jsou Rails postaveny na jazyku Ruby lze používat balíčkovací systém RubyGems, kde je velké množství již existujících řešení, které jsou dostupné jako knihovny[1].

4.1.2 KOSapi

KOSapi je školní 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. Při vývoji aplikace používám stabilní verzi 2 API.

Z této služby aplikace čerpá hlavně data předmětů a osob v KOSu. Pro připojení ke KOSapi používám již existující knihovnu napsanou v Ruby ze školního projektu VyVy[17].

[1] v 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 aplikaci a u dalších aplikací už nemusí zadávat svoje přihlašovací údaje.

Tuto službu používám pro autentizaci uživatelů do systému. Pro zprovoznění aplikace FELid je nutné splnit technické požadavky, které jsou vystaveny na oficiálních stránkách [8].

4.1.4 Aplikační server

Pro zprovoznění aplikace do reálného provozu jsem použil webový server Apache HTTP server 2.4 [20]. Tento aplikační server jsem musel zvolit, abych mohl splnit dva obecné požadavky aplikace. Aplikace musí být napsaná v Ruby on Rails a využívat aplikaci FELid pro autentizaci. Tato verze webového serveru totiž dovolí nainstalovat zásuvné moduly 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 databázovém systému. Tuto vlastnost používám při vývoji a testování aplikace, kde využívám jednoduchý a snadno konfigurovatelný databázový systém SQLite [21], který pro tyto účely bohatě postačí. 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 implementaci 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 komponent[2] 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.

[2] models, 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 atributů 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 známé pod označením CRUD[3]. Tyto metody jsou implementovány pomocí odpovídajících metod HTTP protokolu. Jednotlivé metody rozeberu na příkladech pro zdroj *observations*[4].

[3] create, retriece, update a delete

[4] zdroj aplikace pro práci s hospitacemi

Create je požadavek, který pomocí metody POST vytvoří nový záznam. Příklad pro vytvoření nové hospitace.

```
POST /observations
```

Retrieve je požadavek pro přístup ke zdrojům. Funguje stejným způsobem jako běžný požadavek na stránku pomocí GET metody. První příklad vrátí seznam všech hospitací. Druhý příklad vrátí podrobnosti o konkrétní hospitaci.

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

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

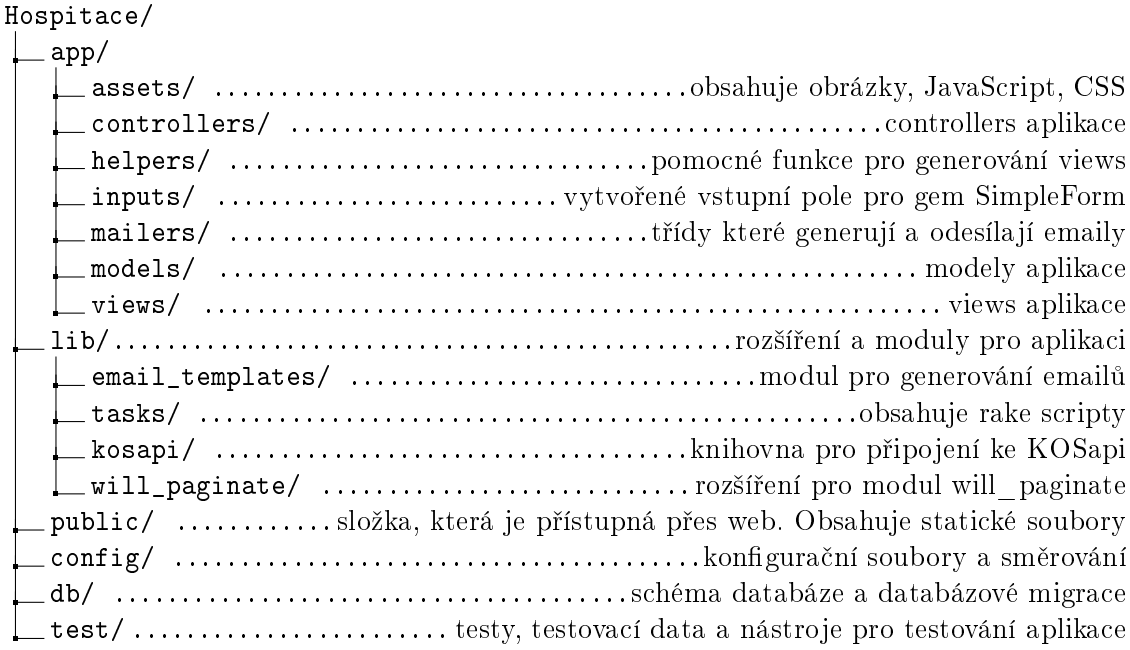
```
PUT /observations/1
```

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 generátor v Ruby on Rails. Proto nepopíšu celou strukturu, ale pouze části aplikace, které jsou nejdůležitější pro její vývoj. Na obrázku 4.1 je popsaná strukturu nejdůležitějších složek a k nim popis obsahu.



Obrázek 4.1: Struktura aplikace

Kapitola 5

Realizace

Po dohodě s vedoucím práce byl zvolen iterační vývoj aplikace. Za účelem postupného vyvíjený 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 nejobsáhlejším ze všech iterací. Jedním z důvodů bylo seznámení se s novou technologii Ruby on Rails. První iterace měla za cíl navrhnout a vytvořit základní architekturu aplikace s napojením na KOSapi viz. 4.1.2 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 a databázi aplikace, bylo potřeba nejdřív vyřešit jak se připojit ke KOSapi. V této části vycházím z prototypu aplikace pro správu hospitací. Ta využívá již naprogramovanou knihovnu z projektu VyVy [16]. Poté bylo potřeba vytvořil modely tak, aby umožnily komunikaci mezi KOSapi a databází aplikace. Při implementování knihovny do aplikace jsem musel vyřešit lokalizaci jazykových konstant v datech získaných z API. Řešení bylo jednoduché, stačilo rozšířit knihovnu 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 po dobu vývoje 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 jedena z kritických oblastí, kterou bylo potřeba vyřešit hned na začátku vývoje. V aplikaci potřebuji autorizovat uživatele jak podle role, tak i podle vztahu k hospitaci. Proto jsem hledal modul, který by dokázal nadefinovat pravidla autorizace. Knihovna, kterou jsem použil, se jmenuje CanCan [4] . Tato knihovna má velmi jednoduchý, ale i přesto flexibilní zápis pravidel. Tyto pravidla umí filtrovat jak podle zdroje, tak i podle jednotlivých instancí záznamů. Umí také dokáže filtrovat metody v kontrolorech i celé zdroje aplikace. Veškerá pravidla jsou definována na jednom místě[2] aplikace.

Příklad zapsaného pravidla pro administrátora hospitací pomocí knihovny CanCan ve třídě modelu Ability. První pravidlo `can` umožní zobrazovat, vytvářet, upravovat a mazat ze zdroje Observation. Zároveň druhé pravidlo `cannot` zakáže operace pro správu záznamů, které administrátor hospitací 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 uživatelé uchovávám v modelu Role. Kde jednotlivé role uživatele ukládám do jednoho atributu `roles_mask` pomocí bitové masky. V bitové masce jsou reprezentovány role čísly mocniny 2. Díky tomu lze skládat role pomocí bitové operace OR. V tabulce 5.1 je přehled rolí s číslem reprezentující bitovou masku pro roli.

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

Tabulka 5.1: Reprezentace rolí v bitové masce

[2] model Ability

5.1.2.5 Uživatelské prostředí

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

Pro usnadnění implementace knihovny Bootstrap do aplikace jsem použil další dvě knihovny. První je modul SimpleForm [15], který 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 byla možnost upravovat formuláře bez zásahu do zdrojového kódu aplikace. Dalším cílem bylo třeba vyřešit problém s KOSapi, který vznikl při přechodu na nový semestr. Služba neudržuje data instancí předmětů z minulých semestrů a proto 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 dále ve vývoji. Musel jsem proto předělat datovou část. Rozšířil jsem databázi o entity z kapitoly 3.3 tak, aby data z KOSapi byla uložena v databázi. Data je potřeba synchronizovat v databázi s KOSapi. O synchronizaci se starají *rake*[3] scripty, které přidávají nové záznamy nebo je aktualizují. Synchronizační scripty se spouštějí každý den pomocí programu *Cron*[4].

Po přidání nových entit bylo potřeba na-implementovat asociace mezi novými entitami. V modelu KOSapi je spousta asociací mezi entitami osoba - paralelka a osoba - instance předmětu. V aplikaci jich celkem používám šest. Všechny tyto asociace[5] mají kardinalitu N:M. Pokud bych použil klasickou dekompozicí přes pomocnou tabulku, která rozloží asociace na 1:N a 1:M. Vzniklo by mi šest pomocných tabulek. Proto jsem se rozhodl použít jiný způsob dekompozice přes polymorfní asociace [9] v Ruby on Rails. Výhodou tohoto řešení je

[3] rake je sada nástrojů používaná pro vývoj a nasazení Rails aplikací
[4] Cron je program, který na pozadí operačního systému spouští naplánované úlohy
[5] asociace garant, přednášející, vyučující, instruktor a zkoušející.

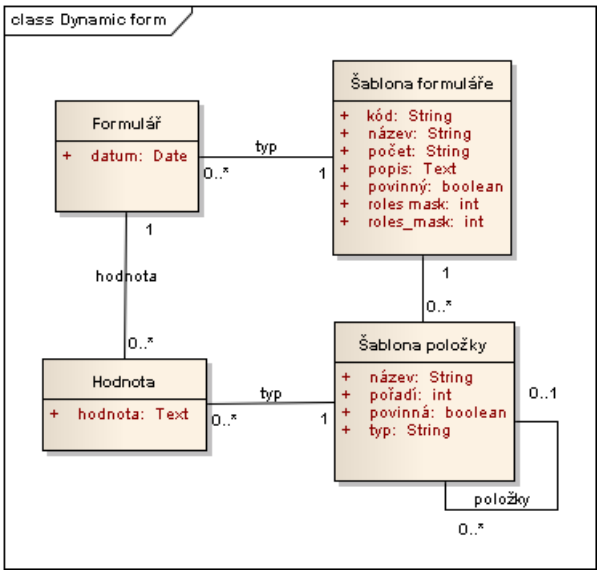
redukce počtu pomocných tabulek. Polymorfní asociace zredukoval počet ze šesti na jednu tabulku. V tabulce 5.2 jsou popsány jednotlivé atributy pomocné tabulky *PeopleRelated* s příklady hodnot.

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

Tabulka 5.2: Popis atributů tabulky 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. Tato šablona definuje základní vlastnosti: minimální, 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 a jiné. Na obrázku 5.1 je doménový model dynamický formulářů.



Obrázek 5.1: Doménový model dynamických formulářů

Hodnotící formuláře se generují podle šablony formuláře získané z databáze. Šablona je reprezentovaná modelem *FormTemplate*. Obsahuje veškeré informace potřebné pro formulaci 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 popisuji v sekci hodnocení výuky 2.3.2.5. Rozdělení formulářů do skupin s kombinací data vytvoření využívám pro verzování šablon. Šablony musím verzovat, jinak by mi při úpravě nebo smazání existující šablony, mohla nastat ztráta dat z vyplněných formulářů.

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

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

2. 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.

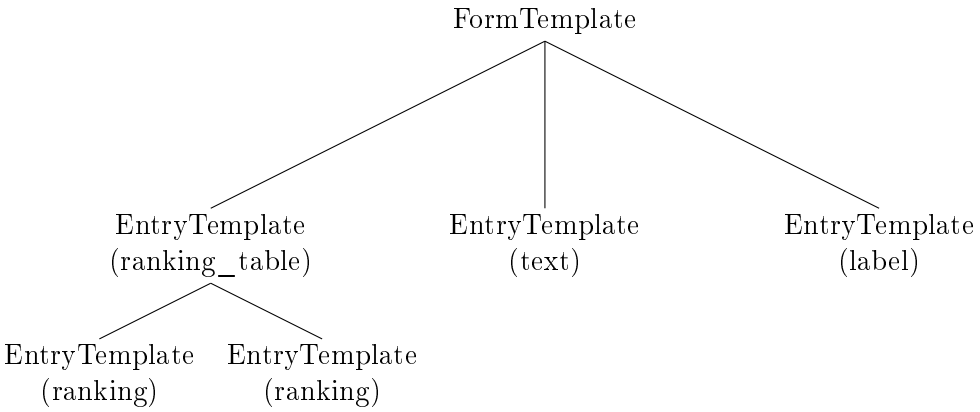
3. Hospitující i hospitovaný může vyplnit maximálně jeden formulář.

U dynamických formulářů jsem musel také řešit jejich chování pro různé role v systému. Proto jsem přidal do šablony dva atributy s bitovými maskami rolí jedna maska pro role, které mohou zobrazit formuláře, druhá maska pro vyplnění formuláře. Používám stejný způsob rozlišování rolí v bitové masce jako u rolí pro uživatele viz 5.1.2.4.

Typ	**Návratová hodnota**	**Popis**
label		textový popisek
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 zadávaní známkování od A do F
ranking_scale		tabulka s hodnotící stupnicí
note	text	vstupní element pro napsání textové poznámky

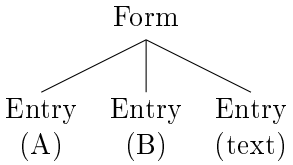
Tabulka 5.3: Seznam typů elementů

Obsah formuláře generuji pomocí položek, ty jsou reprezentovány modelem *EntryTemplate*. Položky formuláře definují co se má vykreslit a kde. Umístění jednotlivých položek v dokumentu je definované ve stromové struktuře 5.2, kde kořenem stromu je šablona formuláře a ta se postupně větví přes potomky. 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*.



Obrázek 5.2: Příklad stromová struktura formulářů

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 uložené v *Entry* 5.3.



Obrázek 5.3: Příklad uložených dat

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 automatického zálohování databáze.

5.3.2 Postup

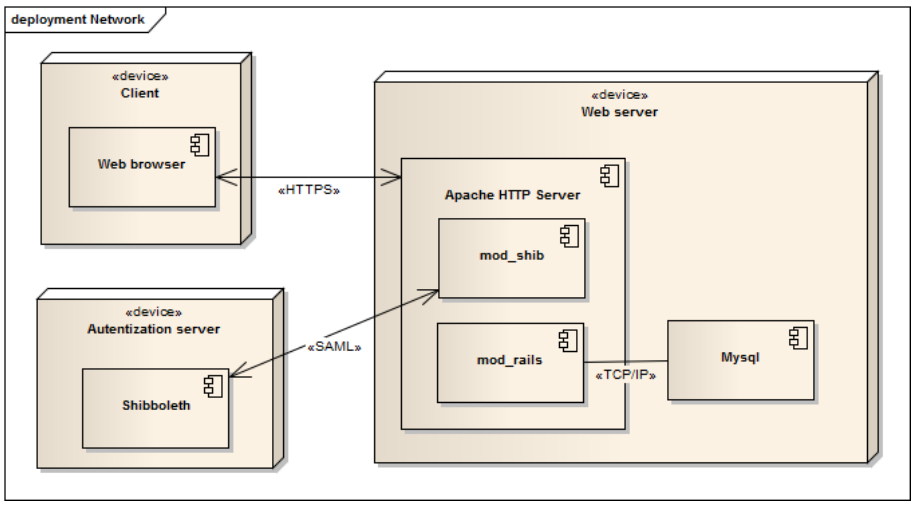
5.3.2.1 Nasazení aplikace

Na diagramu nasazení 5.4 jsou znázorněny komponenty potřebné pro rozjetí aplikace. Protože na serveru byl nainstalován 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]. Tento program dovolí nainstalovat různé platformy Ruby na jednom počítači. Další přínosem je spravování gemsets[6], díky nimž lze zprovoznit několik aplikací zároveň na jednom počítači. Do nainstalovaného webového serveru bylo potřeba ještě přidat zásuvný modul, který umožní nasadit Rails aplikace. Modul se jmenuje Passanger (mod_rails) a instaloval jsem ho do Apache podle návodu na oficiálních stránkách [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*. Při konfiguraci programu jsem postupoval podle návodu, který se nachází na wiki stránkách FELid [7]. Po splnění všech požadavků stačilo jen zažádat o registraci aplikace.



Obrázek 5.4: Diagram nasazení

5.3.2.3 Zálohavání databáze

Pro automatické zálohování databáze jsem použil existující řešení Ruby aplikace *Backup* pro zálohování databází. 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 externí uložiště[7]. Záloha se spouští pravidelně pomocí `Cron` každý den brzy ráno příkazem:

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

[6] balíky knihoven mezi kterými lze přepínat pro různé aplikace
[7] pro příklad Amazon Simple Storage Service (S3)

5.3.3 Výstup

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

5.4 Čtvrtá iterace

5.4.1 Zadání

Tato iterace je poslední a proto většinu zadání tvořily připomínky od zadavatele. V této iteraci patří mezi hlavní úkoly vytvoření modulu, který bude generovat emailové zprávy po vyplnění hodnotícího formuláře a integrovat autentizaci přes FELid.

5.4.2 Postup

5.4.2.1 Autentizace

V této fázi vývoje byla aplikace zaregistrována ve FELid. Před integrováním autentizace přes FELid, je potřeba otestovat aplikaci. Nejdůležitější je otestovat správnou funkčnost autorizace u všech rolí v celém systému. Pro zprovoznění autentizace přes FELid není možné jednoduše testovat aplikaci pod různými uživateli. Od zprovoznění nebudu mít totiž kontrolu nad autentizací.

Integrace do aplikace je pak velmi jednoduchá. Stačilo jen odstranit dočasný způsob autentizace, který jsem implementoval v první iteraci. Aplikace FElid poskytuje mé aplikaci informace o přihlášeném uživateli prostřednictvím atributů v requestu `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. Šablona se skládá z předmětu, textu zprávy a názvu 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 z hospitací, tak jsem vytvořil knihovnu `EmailTemplates` pro generování obsahu 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 nastavit podporované značky. Největším zádrhelem v generování obsahu zprávy je získat data z hospitací. Vstupem do generátoru je pouze instance šablony a hodnocení hospitace. Některá data je potřeba získat přes několik relací mezi modely.

Knihovna obsahuje dva moduly. První částí je modul `ModelHelper`, který slouží pro rozšíření libovolné třídy modelu v aplikaci. Rozšířenému modelu umožní pomocí metody `attrs_tagged(*args)` povolit atributy ze kterých bude generátor získávat data.

Příklad jednoduchého nastavení modelu `Person`, které umožní generátoru 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 slouží pro vytvoření vlastního generátoru pomocí jednoduchého nastavování. Modul poskytuje metodu `source(model,name,*path)` pro nastavení zdrojů dat. Metoda využívá k nastavení generátoru nastavení z rozšířeného modelu, který byl rozšířen modulem `ModelHelper`, název zdroje a cestu k instancím modelu s daty. Cesta je definována posloupností asociací mezi modely.

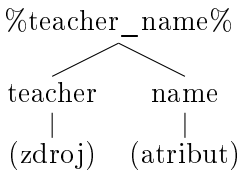
Příklad implementace jednoduchého generátoru `EmailBuilder`, který dokáže vkládat do emailů informace o hospitovaném učiteli.

```
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 5.5 se skládá ze dvou základních částí, ze jména zdroje a jména atributu, ze kterého generátor získá data pro nahrazení značky.

Výhodou této knihovny je krátký a jednoduchý zápis pro nastavování vlastního generátoru, který dokáže přes asociace mezi modely získávat data. V aplikaci jsem celkem nadefinoval pro devět zdrojů 32 značek.



Obrázek 5.5: Struktura značky

5.4.2.4 Odesílání emailů

Dle zadání stačilo implementovat odesílání zpráv po akci vytvoření hodnotícího formuláře. Podle druhu hodnotícího formuláře se vygeneruje zpráva a odešle se administrátorovi 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 kvalitavyuky.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 automaticky pomocí testovacích nástrojů v Ruby on Rails. Aplikace byla testována v průběhu celého vývoje.

6.1 Automatické testování

Automatické testy je důležité používat na místech, kde je velké riziko vzniku chyb. Chyby nejčastěji vznikají při přidávání, nebo úpravě funkcionalit v aplikaci. Pro automatické testování jsem použil testovací nástroje v Ruby on Rails. V Ruby on Rails se testy dělí do tří kategorií Unit, Functional a Integration.

6.1.1 Testovací data

Pro automatické testování je potřeba připravit testovací data. Testovací data jsou obsažená 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 různé činnosti v jednotlivých controllerech aplikace. Controllery zpracovávají příchozí webové requesty a nakonec vyrendrují view. Tento typ testů testuje:

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

- Byli jsme úspěšně přihlášeni?
- 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. Tyto testy se používají k testování větších celků v rámci aplikace.

6.2 Manuální testování

Manuálně jsem testoval části aplikace u kterých se špatně vytvářejí automatické testy, nebo byl potřeba lidský úsudek[1]. 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.

[1] testování uživatelského prostředí

Kapitola 7

Závěr

Výstupem mé práce je aplikace pro evidenci hospitací. Aplikace je nasazená na serveru 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. Aplikace používá dvě fakultní aplikace. Pro autentizaci používá aplikaci FELid, což 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 následně byli zdokumentovány. 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 v čas aplikaci nasadit do reálného provozu.

Tato práce mi přinesla spoustu zkušenosti jak s novou technologií Ruby on Rail, kterou jsem do té doby nepoužil, tak i s použitím externích aplikací.

7.1 Možnosti v pokračování práce

Možností pokračování v práci je několik. V první řadě 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 provedl 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>.

[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 HIPP. *SQLite* [software]. Dostupné z: <http://www.sqlite.org/download.html>.

[22] HLAVÁC, 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] KRĘŻELOK, 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 Retriece 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

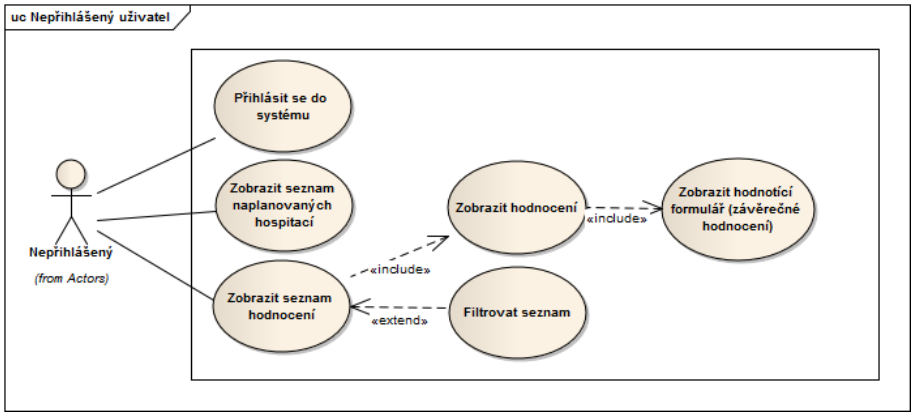
WWW WorldWideWeb

YAML Ain't Markup Language

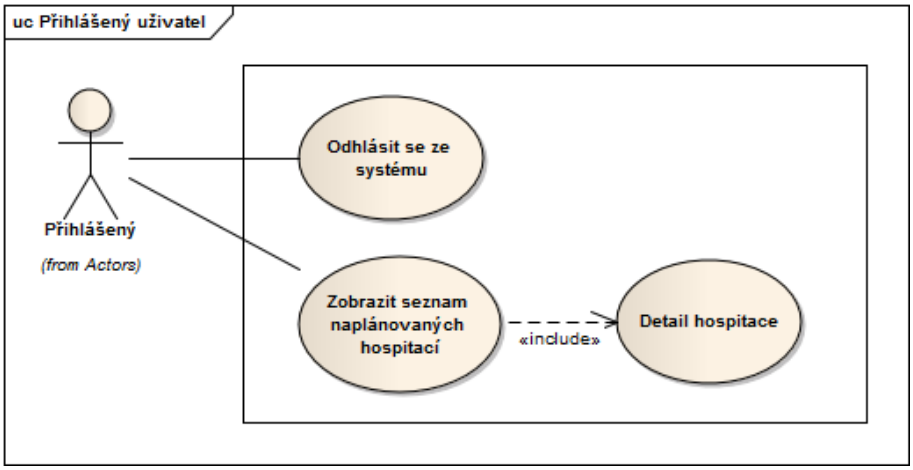
Příloha B

UML diagramy

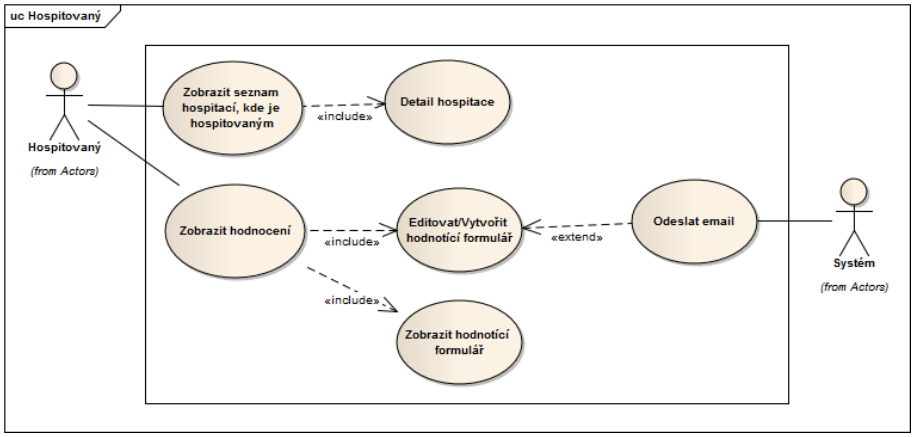
B.1 Use cases



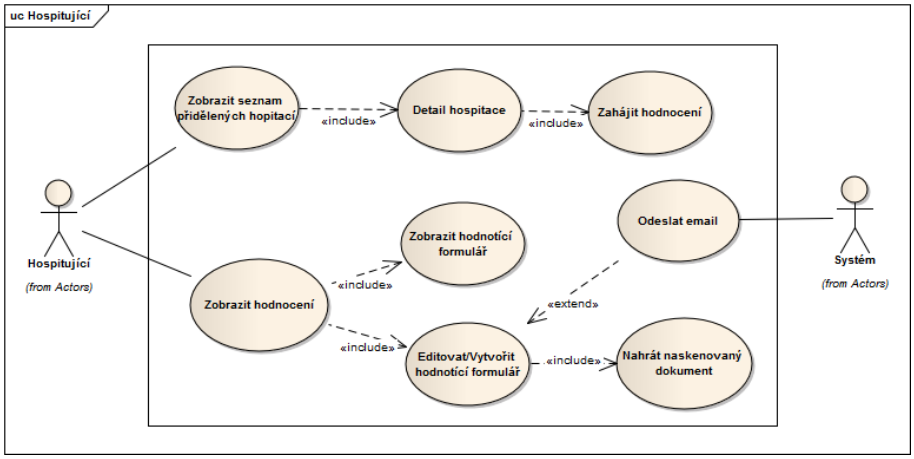
Obrázek B.1: Use case - nepřihlášený uživatel



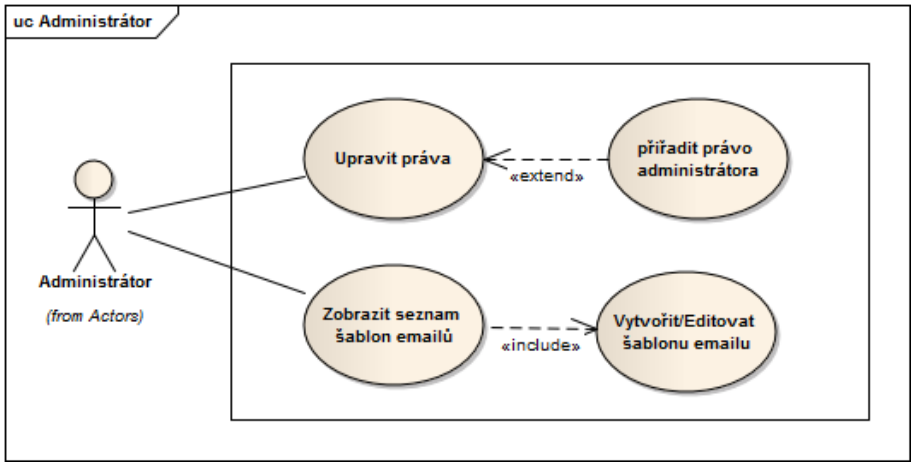
Obrázek B.2: Use case - přihlášený uživatel



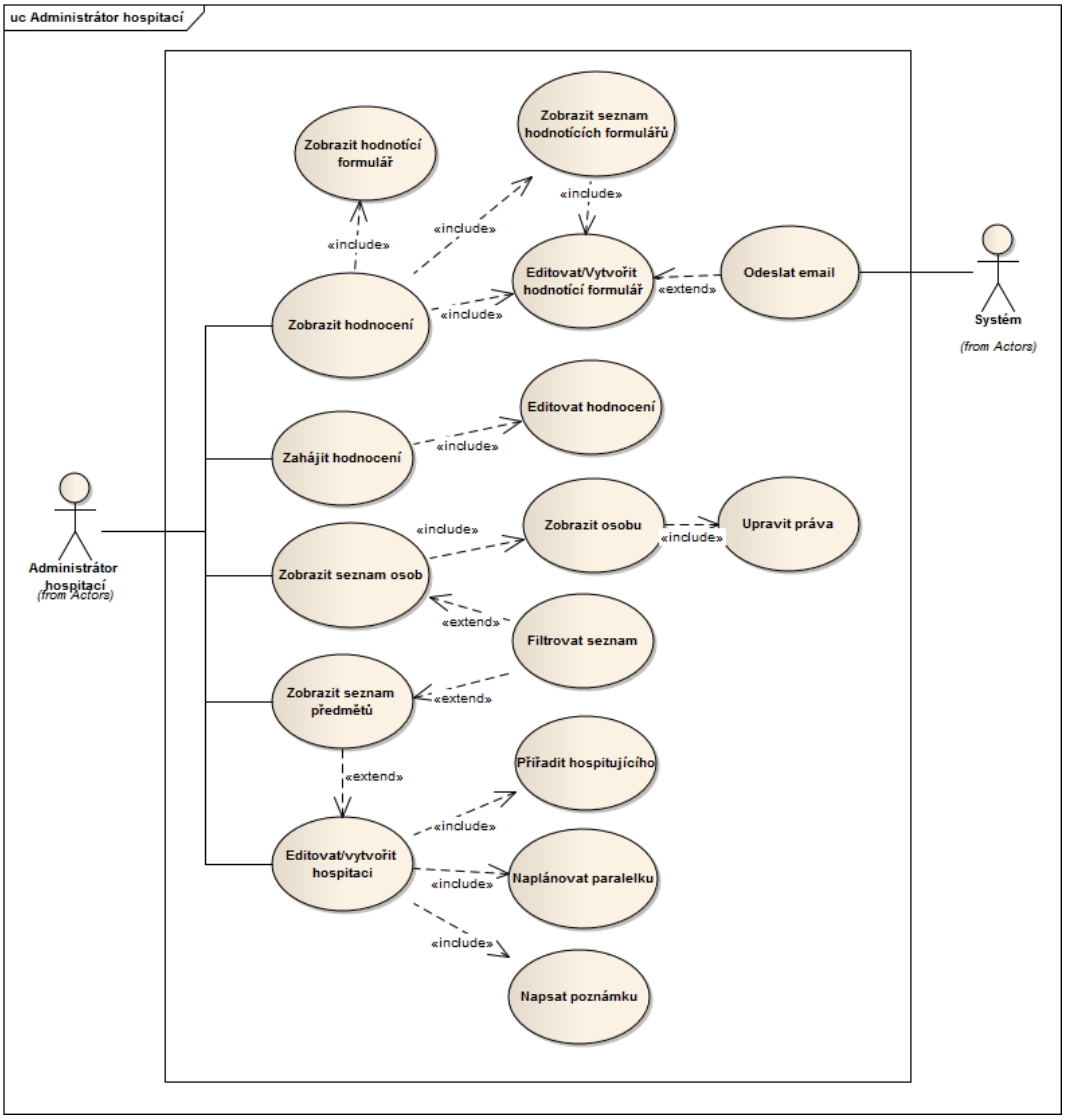
Obrázek B.3: Use case - hospitovaný



Obrázek B.4: Use case - hospitující



Obrázek B.5: Use case - administrátor



Obrázek B.6: Use case - administrátor hospitací