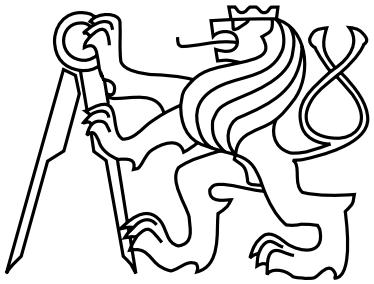
Systém pro evidenci hospitací v Ruby on Rails

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

12. 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 používaná 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 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.

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 [23]. Systém je postaven frameworku Ruby on Rails, který je určen pro vývoj moderních webových aplikací.

[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. 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 [23]. 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 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 [21], který definuje jak se

mají hospitace provádět. A druhým zdrojem je prototyp aplikace z rozpracované bakalářské práce Daniela Krężeloka Návrh a implementace systému pro správu hospitací [24].

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 a návrh aplikace. Hlavně na začátku mi velmi pomohl pochopit problematiku hospitací a velmi užitečné informace pro samotný vývoj mé aplikace. Horší to bylo se samotnou aplikací. Verze programu kterou jsou získal nebyla dodělaná jelikož se mi ji nepodařilo zprovoznit tak aby byla funkční. Po hlubším zkoumání a upravování zdrojových kódu jsem objevil co aplikace umí. Bylo na implementované plánování hospitací a k nim šablony hodnotících formulářů. Další co prototyp umí je získat data přímo z KOSapi 4.1.2.

Důvod proč navazuji na práci Daniel Krężelok je ten že 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 popisuji hlavní procesy při vykonávání hospitací. Tyto informace čerpám z již zmíněného dokumentu Postupy pro kontrolu kvality výuky [21].

2.3.2.1 Administrátor hospitací

Administrátorem hospitací zajišťuje plánování a řízení hospitací pro studijní obor ke kterému je přiřazen. Administrátora hospitací je jedna osoba určená garantem studijního oboru.

2.3.2.2 Plánování hospitací

Pro naplánování hospitace musí administrátor hospitací definovat předmět,vyučovací hodinu a datum hospitace. Hospitace může probíhat jak na přednášky tak i na cvičení. Je potřeba i přiřadit hospitující[1], kteří provedou kontrolní návštěvu ve výuce.

Administrátor musí také určit typ hospitace. Existují tři druhy z hlediska vyučujícího:

- Předem ohlášené na konkrétní datum - u tohoto typu hospitace jsou veřejné všechny informace o naplánované hospitaci. Proto je potřeba, aby administrátor naplánoval hospitaci s předstihem a informoval o tom vyučujícího.
- Předem ohlášené bez konkrétního termínu - tento typ hospitace na rozdíl od předešlého typu nemusí mít předem pevně stanovený datum hospitace. Hospitovaný ví o hospitaci, ale neví kdy bude.
- Předem neohlášené - u tohoto typu hospitace je předem informován pouze garant, zástupce garanta a administrátor výuky.

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

2.3.2.3 Provedení hospitace

Kontrolní návštěva předmětu se provede dle naplánování hospitace a výstupem návštěvy z hodiny je dokument Hodnocení výuky, který dokumentuje průběh hodiny a její hodnocení.

2.3.2.4 Hodnocení výuky

Po hospitaci se následuje hodnotící část. Ta se skládá ze čtyř druhů dokumentů, které hodnotí proběhlou výuku. Všechny dokumenty musí být předány administrátorovi hospitací. Ten je vystavení na privátní části webových stránek Rady programu [23]. Po sepsání a vystavení všech dokumentů je hospitace ukončená.

A Hodnocení výuky při hospitaci - ten dokument je písemný výstup z hospitace, který slouží k popisu průběhu výuky. Skládá se z dokumentační části a z hodnotící části. Tento formulář musí vyplňují všichni hospitující.

B Slovní hodnocení hospitační návštěvy hospitujícím(mi) - tento dokument sepíše 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. Sepíše ho jeden z hospitujících. Dokument slouží jako výstup z hospitace a obsahuje klady, zápory, navržená opatření a závěr. Tento dokument je veřejný proto je potřeba ho vytavit i na volně přístupných stránkách.

Kapitola 3

Analýza

Tato kapitola pojednává o analýze a návrhu vhodného řešení aplikace. Výstupem této analýzy jsou funkční a obecné požadavky. Dále návrh a popis domén a nejdůležitější případy užití s aktéry.

3.1 Požadavky

Požadavky na systém se dělí na dvě sekce: obecné a funkční požadavky. Pro definování těchto požadavků jsem vycházel z oficiálního zadání práce tak i z prototypu aplikace, protože mi přesně definuje návrh aplikace a pro implementaci systému i potřebné technologie.

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 FELid.

3.1.2 Funkční požadavky

Tato sekce se zabývá požadavky na funkčnost systému.

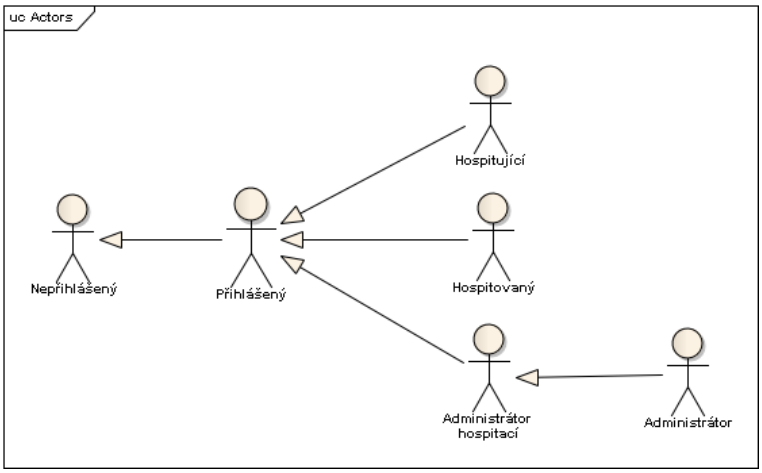
1. Systém umožní spravovat uživatele.
2. Systém umožní průběžné plánování hospitaci.
3. Systém umožní hospitujícímu i hospitovanému prohlížet hospitace [24].
4. Systém umožní vystavit závěrečné hodnocení na veřejné části aplikace [24].

5. Systém umožní hospitovanému sepsat stanoviska k názorům hospitujícího [24].
6. Systém umožní hospitujícímu nahrát naskenovaný dokument hodnocení výuky [24].
7. Systém umožní hospitujícímu napsat slovní hodnocení z výuky [24].
8. Systém umožní hospitujícímu napsat závěrečné shrnutí hospitace [24].
9. Systém bude odesílat emailem zprávy o vyplnění hodnotícího dokumentu příslušným osobám [24].
10. Systém umožní vyhledávat předměty z KOSapi.
11. Systém umožní vyhledávat osoby z KOSapi.
12. Systém umožní upravovat strukturu hodnotících dokumentů.
13. Systém umožní spravovat emailové šablony k hodnotících dokumentů.
14. Systém umožní generovat emailové zprávy ze šablon.
15. 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 jednotlivý aktéři a jejich zobecnění. V další části této sekce rozeberu jednotlivé role a k ním případy užití.

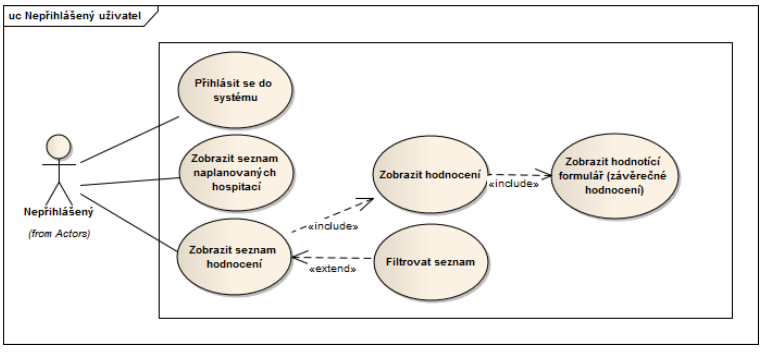
Use cases neboli případy užití je nástroj pro popsání chování jak by systém měl spolupracovat s koncovým uživatelem. Popisuje všechny způsoby jak uživatel komunikuje se systémem.



Obrázek 3.1: Aktéři

3.2.1 Nepřihlášený uživatel

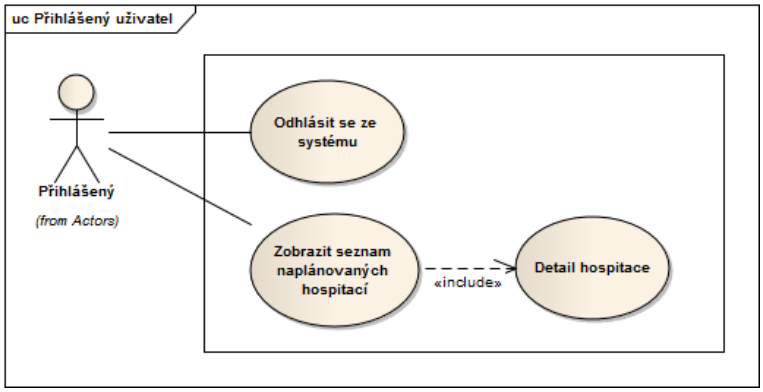
Nepřihlášený uživatel je role pro hosty naší aplikace. V systému má ze všech rolí nejmenší pravomoc. V tomto stavu je každý uživatel, který se doposud nepřihlásil do systému.



Obrázek 3.2: Use case - nepřihlášený uživatel

3.2.2 Přihlášený uživatel

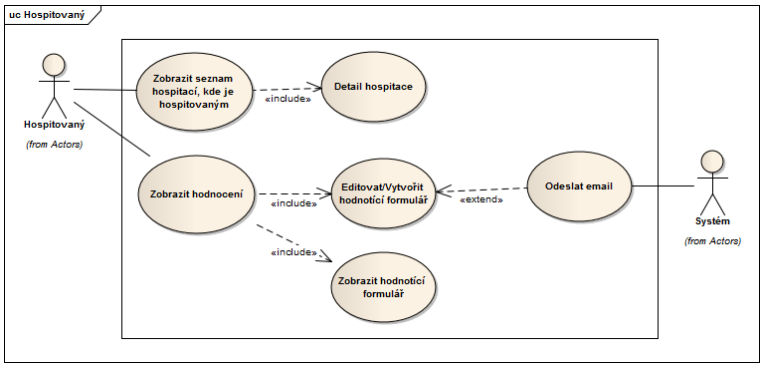
Přihlášený uživatel vychází z role nepřihlášeného uživatele. Je to uživatel, který se do systému přihlásil. Jedná se o základní roli pro všechny další role, které ji rozšiřují.



Obrázek 3.3: Use case - přihlášený uživatel

3.2.3 Hospitovaný

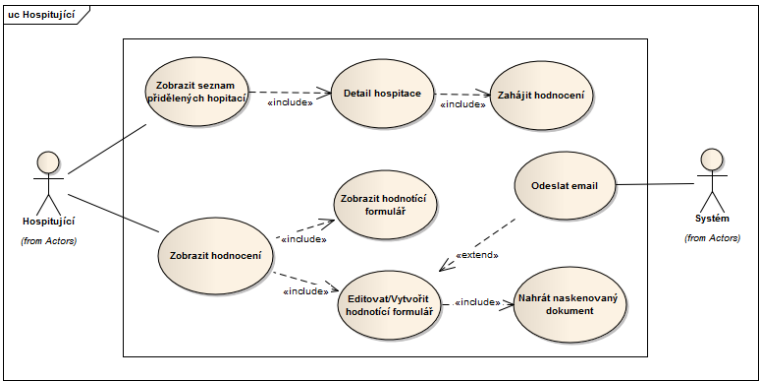
Hospitovaný je role pro přihlášeného uživatele v systému. Je přidělena pro každého vyučujícího, který vyučuje předmět, na němž byla naplánovaná hospitace a proběhla.



Obrázek 3.4: Use case - hospitovaný

3.2.4 Hospitující

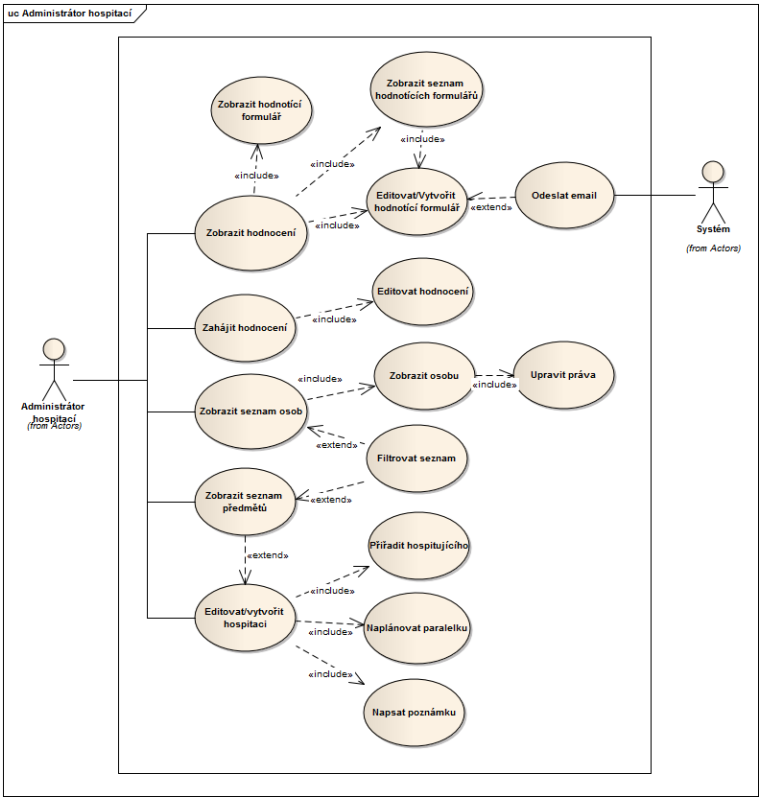
Hospitující je role pro přihlášeného uživatele v systému. Tato role se přiděluje automaticky z naplánovaných hospitací, nebo ji může přidělit administrátor hospitací.



Obrázek 3.5: Use case - hospitující

3.2.5 Administrátor hospitací

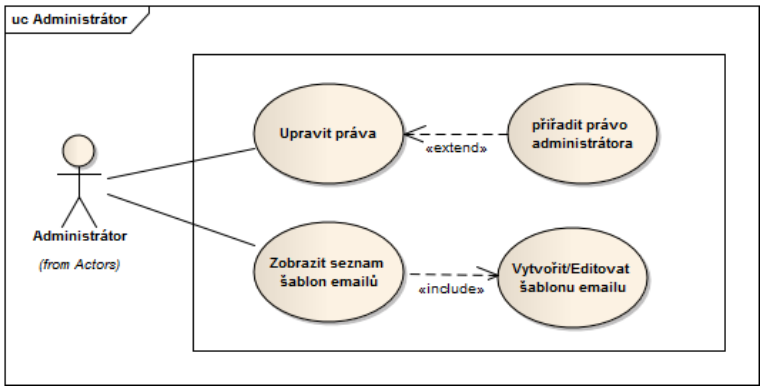
Hlavním úkolem této role je plánovat hospitace na předměty a posléze je spravovat.



Obrázek 3.6: Use case - administrátor hospitací

3.2.6 Administrátor

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



Obrázek 3.7: Use case - administrátor

3.3 Doménový model

Doménový model na obrázku 3.8 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

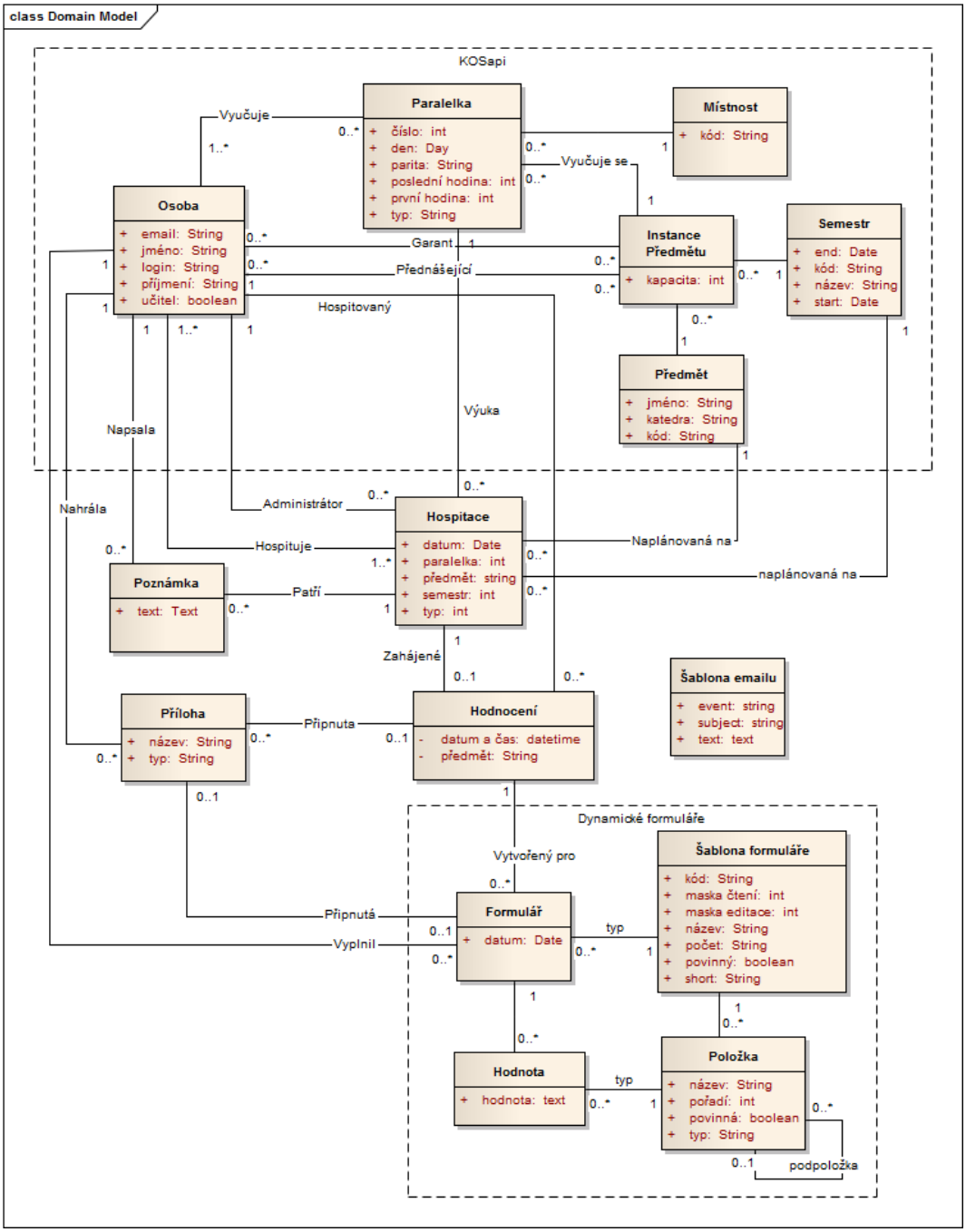
- Osoba - je osoba v KOSu. Každá osoba může být učitelem a studentem.
- Semestr - semestr vyučovaný na FEL.
- Předmět - předměty vyučované na FEL.
- Instance předmětu - jsou instance předmětu vypsané v konkrétním semestru.
- Paralelka - je vypsaná rozvrhová paralelka pro instanci předmětu.
- Místnost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

3.4 Životní cyklus hospitace

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



Obrázek 3.8: Doménový model

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.

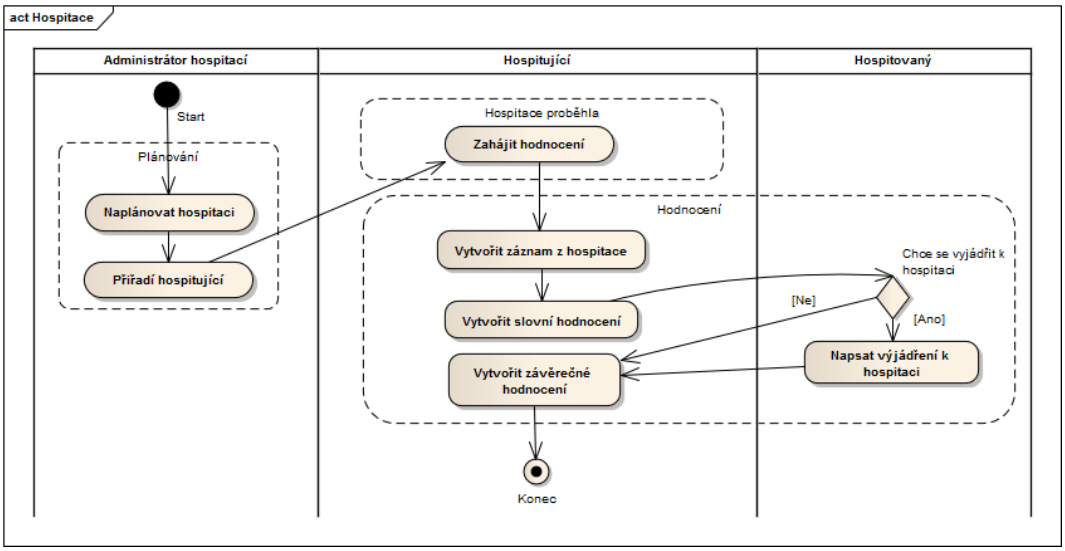
V druhé fázi jeden z hospitujících sepíše slovní hodnocení hospitační návštěvy.

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 cyklu.

3.4.4 Ukončená

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



Obrázek 3.9: 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 [13], zkráceně Rails, je jedno z implementačních omezení, které se nachází přímo v zadání práce. Je to framework primárně určený pro tvorbu webových aplikací napojených na databázi. Framework je postavený na skriptovacím interpretovaným programovací jazyku Ruby [12]. Rails je postaven na návrhovém vzoru model-view-controller viz. 4.2.1. Tento framework používá dva hlavní principy. Prvním principem je Convention over Configuration viz. 4.2.3 a druhým je Don't Repeat Yourself viz. 4.2.2.

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 [16].

4.1.3 FELid

FELid [6] 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. Abych mohl používat v aplikaci FELid je nutné splnit technické požadavky, které jsou napsány na stránkách FELid [7].

4.1.4 Aplikační server

Pro zprovoznění aplikace do reálného provozu jsem použil webový server Apache HTTP server [19] ve verzi 2. Tento aplikační server jsem zvolil kvůli obecným požadavkům aplikace pro využití FELid a pažadavku aby aplikace byla napsaná v Ruby on Rails. Tato verze webového serveru totiž umožňuje instalaci zásuvných modulů Passenger [25] a Shibboleth [22]. 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 [20], který pro tyto účely bohatě postačuje a není potřeba jej složitě konfigurovat. Pro samotné nasazení aplikace do provozu už používám databázový systém MySQL [18].

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) [10] je softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent[1] 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.

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.

[1] models, views a controllers

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) [5] 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) [4] 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) [11] 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[2]. Tyto metody jsou implementovány pomocí odpovídajících metod HTTP protokolu. Jednotlivé metody rozeberu na příkladech pro zdroj `observations`[3].

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 hospitace. 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
```

[2] create, retriece, update a delete
[3] zdroj aplikace pro práci s hospitacemi

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 popsaná strukturu 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 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.

pužité nástroje pro realizaci jsem použil vývojové prostředí netbeans + vedení projektu používám google code. Kde se nachází základ projektu.

5.1 První iterace

5.1.1 Zadání

Tato iterace patřila k nejobsáhlejším ze všech iterací. Jedním z důvodů bylo seznámení z novou technologii Ruby on Rails, s kterou jsem neměl zkušenost. První iterace měla za cíl navrhnout a vytvořit základní architekturu aplikace s napojením na KOSapi [9] 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 nejdřív vyřešit jak se připojit ke KOSapi. V této části vycházím z prototypu aplikace pro správu hospitací. Kde využívá již naprogramovanou knihovnu z projektu VyVy [15]. 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 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 modul authlogic, který jsem zprovoznil pomocí návodu [1]. Tento modul 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 jedena 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. Modul, který jsem použil, se jmenuje CanCan [3] . Tento modul má velmi jednoduchý a přesto flexibilní zápis pravidel, které dokáží filtrovat jak podle zdroje tak i podle jednotlivých záznamů, dokáže filtrovat controllery tak i zdroje aplikace. Veškerá pravidla jsou definována na jednom místě[2].

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 uživatelé 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

[2] model Ability

5.1.2.5 Uživatelské prostředí

Pro vytvoření uživatelského prostředí jsem použil již existující modul Bootstrap [2]. 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 [14], slouží k vytváření formulářů. Základním cílem je nadefinovat rozvržení všech formulářů na jednom místě. Druhým modulem je WillPaginate [17] pros stránkován 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 neudržuje 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 uložené v databázi. Data je potřeba synchronizovat v databázi s KOSapi. O synchronizaci se stará `rake`[3] script stará se o přidává a aktualizaci záznamy. Synchronizační script spuštím každý den pomocí programu `Cron`[4].

Po přidání nových entit bylo potřeba na-implementovat asociace mezi novými entitami. V doménového modelu na obrázku 3.8 je spousta asociací mezi osobou - paralelkou a osobou - instancí předmětu. Protože 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 6 pomocných tabulek. Proto jsem rozhodl požít jiný způsob dekompozice,

[3] rake je sada nástrojů používané 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.

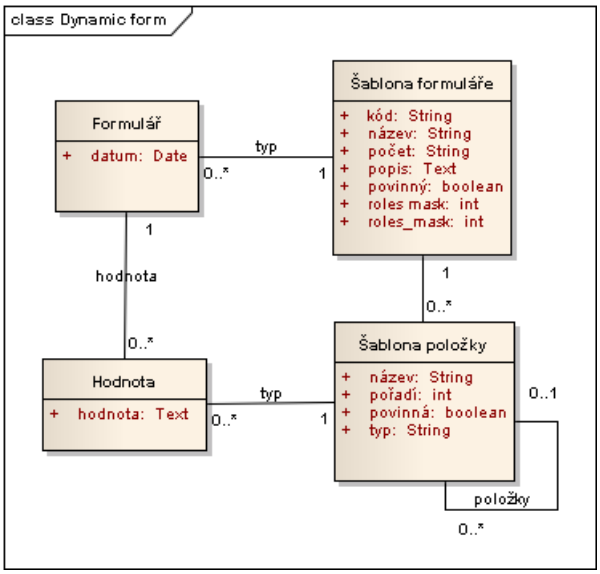
který používá polymorfní asociace [8] v Ruby on Rails. Výhodou této dekompozice je redukce počtu pomocných tabulek. Tento druh asociace zredukoval počet ze 6 na 1 tabulku. V tabulce 5.2 jsou popsány jednotlivé atributy s příklady modelu `PeopleRelated`.

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



Obrázek 5.1: Doménový model dynamických formulářů

Hodnotící formuláře se generuji 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 důležitý vlastnost, definuje skupinu kam patří

formulář. Skupiny formulářů jsou rozděleny podle druhu formuláře a ty popisuji v sekci hodnocení výuky 2.3.2.4. 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 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

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 formululář 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`.

5.2.3 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 Shibboleth pro FELid. Dalším požadavkem bylo zprovoznění automatické zálohování databáze.

5.3.2 Postup

5.3.2.1 Nasazení aplikace

V této fázi bylo potřeba připravit server na nasazení aplikace. Na obrázku 5.2 jde vidět diagram nasazení serveru. Pro podporo rails jsem použil software RVM **??** pro spravování verzí ruby a gemsets.

Do apache jsem musel nainstalovat dva zásuvné moduly. Jeden pro podporu rails aplikací Passanger a pro zprovoznění FELid modul Shibboleth. Pro samotnou konfiguraci Shibbolethu jsem postupoval pomocí návodu na stránkách FELid [**?**].

5.3.2.2 Záloha databáze

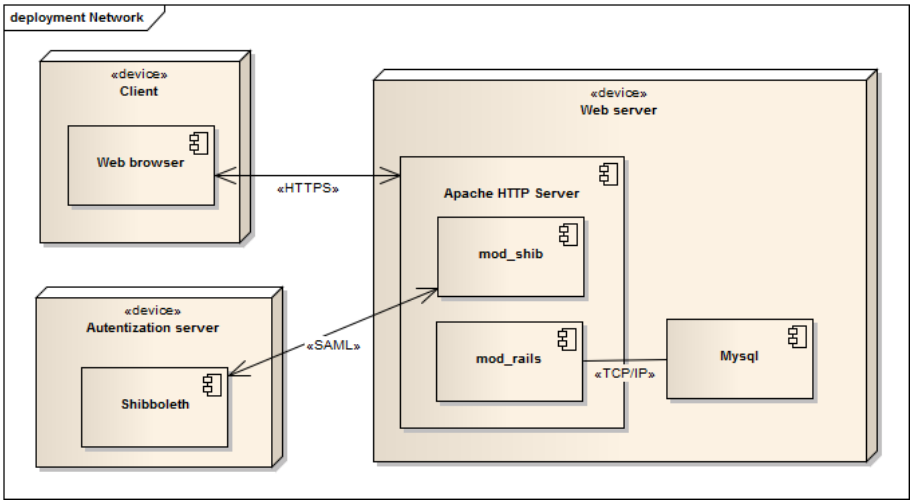
Pro automatické zálohování databáze jsem použil existující ruby aplikaci pro zálohování. Tato aplikace každý den vytvoří zálohu databáze, kterou uloží na disk. Na disku uchovává 300 záloh databáze , při překročení počtu záloh se starší zálohy nahrazují novými. Výhodou tohoto řešení je podpora různých databázových systémů a možnost ukládat zálohy na externích uložiště[6] pro uchování záloh. Záloha se spouští pomocí příkazu v konzoli:

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

5.3.2.3 Cron úlohy

5.3.3 Výstup

[6] pro příklad Amazon Simple Storage Service (S3)



Obrázek 5.2: Diagram nasazení

Kapitola 6

Testování

- Způsob, průběh a výsledky testování.
- Srovnání s existujícími řešeními, pokud jsou známy.

Kapitola 7

Závěr

- Zhodnocení splnění cílů DP/BP a vlastního přínosu práce (při formulaci je třeba vzít v potaz zadání práce).
- Diskuse dalšího možného pokračování práce.

Literatura

[1] *Authlogic: documentation* [online]. [cit. 2012-04-07]. Dostupné z: <http://rdoc.info/github/binarylogic/authlogic>.

[2] *Bootstrap, from Twitter* [online]. 2012-04-07. Dostupné z: <http://twitter.github.com/bootstrap/>.

[3] *CanCan: Getting Started* [online]. 2012. Dostupné z: <https://github.com/ryanb/cancan>.

[4] *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>.

[5] *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>.

[6] *O systému FELid* [online]. [cit. 2012-02-15]. Dostupné z: <http://wiki.feld.cvut.cz/net/felid/about>.

[7] *Požadavky FELid* [online]. 2012-02-15. Dostupné z: <http://wiki.feld.cvut.cz/net/admin/aai/provoz/pozadavky>.

[8] *A Guide to Active Record Associations* [online]. [cit. 2012-04-07]. Dostupné z: <http://guides.rubyonrails.org/association_basics.html>.

[9] *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>.

[10] *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>.

[11] *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>.

[12] *Ruby* [online]. Dostupné z: <http://www.ruby-lang.org/en/>.

[13] *Ruby on Rails* [online]. [cit. 2011-12-15]. Dostupné z: <http://rubyonrails.org/>.

[14] *SimpleForm: Rails forms made easy* [online]. [cit. 2012-04-07]. Dostupné z: <https://github.com/plataformatec/simple_form>.

[15] *Aplikace Vykazování výuky* [online]. 2012. Dostupné z: <https://vyvy.felk.cvut.cz/>.

[16] *Stránky projektu VyVy* [online]. 2012. Dostupné z: <http://code.google.com/p/vykazovani-vyuky-cvut/>.

[17] *Will_paginate: documentation* [online]. 2012-04-07. Dostupné z: <https://github.com/mislav/will_paginate/wiki>.

[18] ORACLE CORPORATION. *MySQL 5.5* [software]. Dostupné z: <http://dev.mysql.com/downloads/mysql/>.

[19] APACHE SOFTWARE FOUNDATION. *Apache HTTP server 2.4* [software]. [přístup 2012-04-07]. Dostupné z: <http://httpd.apache.org/download.cgi>.

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

[21] 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>.

[22] INTERNET2. *Shibboleth SP 2.4* [software]. Dostupné z: <http://shibboleth.internet2.edu/downloads.html>.

[23] KOMÁREK, M. *Kontrola kvality výuky* [online]. [cit. 2012-01-17]. Dostupné z: <https://wiki.feld.cvut.cz/rada_stm/kontrolavyukyverejne>.

[24] KRĘŻELOK, D. Návrh a implementace systému pro správu hospitací. Bakalářská práce, České vysoké učení technické v Praze, Fakulta elektrotechnická, Praha, 2010.

[25] PHUSION. *Passenger* [software]. Dostupné z: <http://www.modrails.com/install.html>.

Příloha A

Seznam použitých zkratek

2D Two-Dimensional

ABN Abstract Boolean Networks

ASIC Application-Specific Integrated Circuit

⋮