Jihočeská universita v Českých Budějovicích
Přírodovědecká fakulta
Ústav aplikované informatiky

Bakalářská práce

Informační systém pro hudební školy

Lukáš Svoboda

Vedoucí práce: Mgr. Miloš Prokýšek, Ph.D.

Studijní program: Aplikovaná informatika

Obor: Aplikovaná informatika

24. dubna 2013

Bibliografické údaje

Svoboda L., 2013: Informační systém pro hudební školy [Information system for music school, Bc. Thesis, in Czech.] - 56p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Abstract:

Subject of the BA thesis is development of information system designed specifically for musical schools. The thesis is divided into several parts. First part is focused on analysis of the issue and on designing a solution. Second part describes development of the system.

The information system is designed with emphasis on future functional and technological extensibility and maintainability which results in SOA product. User interface is a web application composed of several portlets built on top of Liferay portal stack. Web services provider is a standalone application with SOAP interface. User identities as roles are stored in an LDAP server which is utilized by both applications.

Abstrakt:

Tématem práce je tvorba informačního systému pro hudební školy. Práce je rozdělena do dvou částí. První část obsahuje analýzu problému a abstraktní návrh řešení. Druhá část se zabývá implementací a nasazením do provozu.

Informační systém je navržen s důrazem na další funkční a technologickou rozšiřitelnost. Výsledkem je systém s architekturou orientovanou na služby (SOA). Webové uživatelské rozhraní je implementováno pomocí několika portletů běžících v portálu Liferay. Poskytovatelem webových služeb přes protokol SOAP je samostatná aplikace. Obě aplikace používají jako zdroj uživatelských rolí a identit LDAP server.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích, dne 24. 4. 2013

Podpis: .....

Poděkování

Chtěl bych poděkovat především vedoucímu mé bakalářské práce Mgr. Milošovi Prokýškovi, Ph.D. za cenné rady a konstruktivní kritiku. Dále bych chtěl poděkovat Bc. Pavlu Horalovi a Ing. Tomášovi Jarošovi za praktické rady a pomoc s problémy zejména při vývoji. V neposlední řadě bych chtěl poděkovat kantorům Přírodovědecké fakulty za znalosti a dovednosti, kterým mě v průběhu studia naučili a své rodině za podporu a možnost studovat vysokou školu.

Obsah

Kapitola 1

Úvod

Po celém světě funguje na principu franšízy (jedná se o způsob obchodní činnosti, kdy dochází k poskytnutí práva užívat obchodní známku a znalosti vlastněné nějakou korporací[8, p. 401]) síť hudebních škol Yamaha[1] (dále HŠY). V každé škole se vyučuje většinou více než jeden obor (např.: hra na klávesy, flétnu, zpěv,...) v několika učebních programech. Výuka probíhá ve skupinách a vede ji školený kantor.

Pokud se potenciální zákazník rozhodne stát žákem hudební školy Yamaha, musí se zapsat. Může tak učinit na pravidelně pořádaných zápisech anebo se s vedením školy domluví na zápise individuálně. Jakmile je žák zapsán, vzniká mu povinnost uhradit školné a dochází řádně na hodiny.

Administrativní pracovníci škol evidují došlé platby a případně upomínají neplatiče. Dále je potřeba řešit individuální zápisy nových žáků, prodeje not a jiných pomůcek, evidenci docházky apod. Každá škola toto řeší jiným, více či méně efektivním způsobem.

Vzhledem k tomu, jaká je režie se zápisy a platbami a to jak pro zákazníky, tak pro školy, je nasnadě využití informačních technologií k usnadnění klíčových procesů a tudíž zvýšení atraktivity nabízených služeb a snížení nákladů na provoz školy. Vhodné řešení by taktéž mohlo pomoci při marketingových kampaních, v cílení na konkrétní zákazníky, při znovunabízení služeb bývalým žákům či v dalších činnostech, jež jsou v případě primitivních způsobů vedení agendy téměř nerealizovatelné.

Na základě předchozího vznikla myšlenka vytvořit takový informační systém, který by vyřešil uvedené problémy a umožnil v dnešním světě Internetu a počítačových technologií dále se rozvíjet jak jednotlivým školám, tak samotné značce. Tato bakálářská práce se zabývá návrhem a vývojem právě takového informačního systému (dále jen IS).

[1] Hudební školy Yamaha mají celosvětově přes šest milionů záků ve čtyřiceti zemích světa. Značka má více než padesátiletou tradici.

1.1 Cíle projektu

Cílem práce je návrh a implementace IS, který bude podporovat každodenní provoz jednotlivých škol a bude dále rozšiřitelný a upravitelný tak, aby vyhovoval momentálním potřebám organizace.

1.2 Použitá metodologie

Práce se zabývá kompletním vývojem IS od rešerše až po nasazení řešení. Znamená to tyto kroky:

- Analyzovat hudební školy a pochopit podstatu jejich činnosti,
- formulovat funkční a nefunkční požadavky na IS,
- rešerže existujících řešení a jejich vhodnosti pro dané použití,
- navrhnout platformu sloužící jako základ systému,
- implementovat, nasadit a otestovat platformu,
- zahájit iterativní vývoj v duchu agilních metodik (manifest agilního vývoje[2]).

Lze pozorovat, že se dá vývoj rozdělit na dvě logické části. První část odpovídá spíše metodice vodopád, zatímco druhá část probíhá v duchu agilního vývoje inspirovaného metodikami Scrum[3] a Getting Real[4].

[2]Manifesto for Agile Software Development. Principles behind the Agile Manifesto [online]. 2001 [cit. 2013-04-13]. Dostupné z: http://agilemanifesto.org/principles.html

[3]*Scrum Alliance* [online]. 2012 [cit. 2013-04-20]. Dostupné z: http://www.scrumalliance.org/

[4]The smarter, faster, easier way to build a successful web application. *Getting Real* [online]. 2013 [cit. 2013-04-20]. Dostupné z: http://gettingreal.37signals.com/

Kapitola 2

Analýza

Tato kapitola se zabývá analýzou organizace a jejích požadavků. Na počátku analýzy byl distribuován dotazník provozovatelům hudebních škol Yamaha v České a Slovenské republice, který měl zmapovat jejich přístup k agendě a postoj k technologiím. Přestože na dotazník byla ochotna odpovědět pouze určitá část dotázaných, získané informace stačily k sformování představy o školách. Dále následovala řada řízených rozhovorů s provozovateli a zaměstnanci a osobních návštěv HŠY v Českých Budějovicích, Písku, Českém Krumlově a Táboře.

Získané informace umožnily definovat funkční a nefunkční požadavky na systém. Dále byla v rámci dotazování provedena rešerše stávajících řešení.

Několik franšízantů již nějaký internetový systém používá, ať se jedná o již hotovou službu, nebo vlastní silou vytvořenou aplikaci, všichni dotázaní se v zásadě shodují na jednom: Se systémemy nejsou spokojeni, a to hlavně proto, že nenabízejí to, co jejich konkrétní organizace potřebuje a řešení na míru by ocenili.

2.1 Dotazníkové šetření

Dotazníky byly distribuovány pomocí Internetu a elektronické pošty. K vytvoření dotazníkového formuláře byla použita služba Google Drive - Forms. V případě tohoto šetření je možné pozorovat určitou faktickou nepřesnost v získaných datech. Data zadávaná do dotazníku se z určité části lišila u škol, kde byla provedena osobní návštěva a sledování provozu.

Z posbíraných dat byly získány určité znalosti, je ale potřeba počítat s určitou faktickou nepřesností a subjektivní korelací faktů dotazovanou osobou. Mezi jenpodstatnější zjištění patří:

- 73 % škol provozuje vlastní webové stránky.
- Agenda spojená s evidencí žáků je v 60 % vedena v MS Excel a v 67 % ji má na starosti ředitel školy.

- Žádná ze škol neumožňuje platbu školného kreditní kartou přes internet.
- V průměru 45 % dotázaných má utvořen názor na informační systémy.

2.2 Logický Rámec

Popis projektu	**Objektivně ověřitelné ukazatele**	**Prostředky ověření**	**Předpoklady**
Cíl projektu *Zefektivnění administrace hudebních škol, usnadnění komunikace zaměstnanců mezi sebou a zlepšení kontaktu školy se zákazníkem.*	*Nejméně čtyři školy mají zájem o systém. Školy využívající systém jsou efektivnější v administrativě a kromě zvýšeného pohodlí klientů šetří zdroje.*	*Zpětná vazba od provozovatelů škol, jejich zaměstnanců a zákazníků pasivně ve formě dotazníků či aktivně z dotazů a připomínek v systému Redmine.*	
Účel projektu *Využití IT k zautomatizování případně usnadnění určitých business procesů, ke sběru dat a jejich dalšímu využití v marketingových strategiích.*	*Veškerá data jsou uchována v databázi a získání určité informace je rychlejší a jednodušší. Navíc některé procesy probíhají částečně nebo plně automaticky.*	*Srovnání náročnosti administrativních úkonů současného systému se systémem novým.* *Snížení počtu zaměstnanců potřebných k administrativě školy a tudíž snížení nákladů na provoz.*	*Ochota zaměstnanců a zákazníků naučit se a aktivně využívat nový IS.* *Využívání systému bude ze strany zaměstnanců zodpovědné a vkládaná data budou validní.*
Výstupy *- implementační dokumentace* *- aplikační platforma pro vývoj IS* *- dokumentace kódu aplikace* *- uživatelská dokumentace*	*Do 28. 2. 2012 bude spuštěn IS ve stavu umožňujícím testování.* *Do 1. 9. 2013 je systém připraven k produkčnímu spuštění a školy jej mohou využívat. Následovat bude další rozvoj IS. Systém splňuje funkční a nefunkční požadavky.*	*- Data ze systému Redmine* *- Audit - vyhodnocení míry naplnění požadavků* *- Dotazník*	*Implementace a součinnost škol při vývoji bude probíhat bez větších problémů.* *Bude vytvořena speciální skupina uživatelů IS aktivně se podílející na vývoji používáním a připomínkováním systému a tím umožní další zvyšování jeho kvality.*

Činnosti	**Aspekty nutné pro realizaci**		
- Analýza požadavků *- Návrh základů systému* *- Implementace základu systému* *- Nasazení systému na vývojovou instalaci* *Další vývoj v týdenních sprintech až do ukončení projektu:* *- Požadavek na změnu od klienta* *- Analýza a návrh řešení* *- Implementace řešení* *- Předvedení klientovi* *- Aktualizace dokumentace*	*- Schopný tým s potřebnou odborností* *- Přístup k business procesům organizace* *- Součinost klienta se zhotovitelem* *- Vývojové, testovací a produkční prostředí*	*- 09/2012* *- 10/2012* *- 11/2012 - 2/2013* *- 3/2013* *- 3/2013 -*	*Předpokladem je dostatek lidských zdrojů a jejich kvalitně odvedená práce.* *Rizika jsou plnění dílčích částí po termínu, špatná komunikace se zákazníkem a nebo špatná kvalita práce.*

2.3 Požadavky na systém

2.3.1 Funkční požadavky

Podsekce obsahuje detailní výčet funkčních požadavků neboli požadavků na konkretní vlastnosti a funkcionality systému.

- Systém je dostupný z většiny dnes běžně používaných desktopových platforem. Zejména se jedná o Microsoft Windows od verze XP, Apple Mac OS X od verze 10.5.
- Aplikace využívá centrální uložiště, tzn. všichni uživatelé v rámci jednoho kontextu (franšízant) přistupují ke stejným datům.
- Do systému se všichni přihlašují pomocí uživatelského jména a hesla.
- Uživatel má možnost přidat novou osobu do systému.
- Uživatel má možnost zapsat žáka do hodiny.
- Uživatel má možnost vyhledávat v hodinách podle kritérií.

- Uživatel může zobrazit zapsané žáky v jednotlivých hodinách.
- Systém umožňuje zařazovat uživatele do skupin a přihlášený uživatel má právě taková práva, jaká má skupina, do které je přiřazen.
- V systému je možné evidovat vyučované kurzy, programy a jednotlivé hodiny.
- Aplikace umožňuje v jednom okamžiku připojení a vykonávání požadavků více uživatelů najednou.
- Jednu instalaci aplikace může využívat několik nájemců[1] naráz, přičemž jsou jejich data logicky oddělena. Znamená to, že nájemce A nemá přístup k datům nájemce B.
- Aplikace jako celek je centrálně spravovatelná. Znamená to například, že údržba nebo aktualizace probíhají naráz na všech instalacích pro všechny nájemníky.
- Každý nájemce má možnost určité personalizace aplikace.
- Data jsou pravidelně zálohována a nájemník systému si může nechat na požádání obnovit databázi.

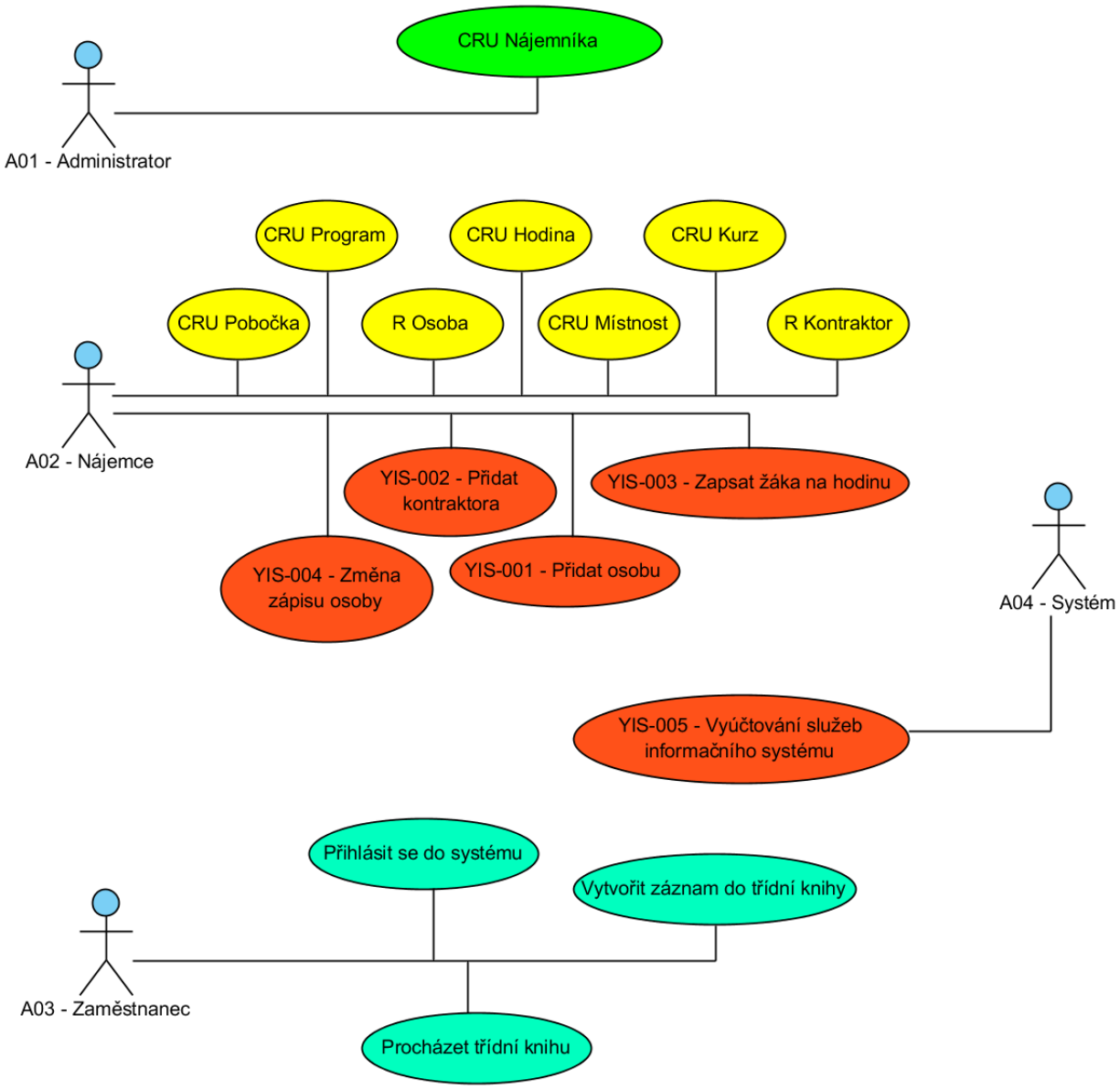
2.3.2 Nefunkční požadavky

- Aplikace je napsaná v jazyce Java.
- Jedná se o webovou aplikaci.
- Systém je škálovatelný.
- Systém splňuje základní požadavky na bezpečnost.
- Aplikace je navržena a strukturována tak, aby umožňovala co nejsnazší vývoj a další úpravy.
- Uživatelské rozhraní je co nejjednoduší s důrazem na pochopitelnost a jednoznačnost jednotlivých ovládacích prvků.
- Kód aplikace je dokumentovaný, dobře čitelný a udržovatelný.
- Systém splňuje požadavky na dostupnost a odezvu.

2.3.3 Případy užití

Na obrázku 2.1 je vyobrazen diagram případů užití. Z diagramu jsou patrné hlavní funkcionality systému a v předprodukční verzi by měl systém tyto případy užití implementovat. Červeně označené případy mají před názvem identifikátor a popsaný scénař. Scénáře k UC se nachází v příloze C Scénáře případů užití.

[1] Nájemcem se rozumí provozovatel hudební školy nebo franšízant, jinými slovy osoba, se kterou byla uzavřena dohoda o poskytování služeb



Obrázek 2.1: Zjednodušený Use Case diagram systému.

Kapitola 3

Návrh

3.1 Použité technologie

3.1.1 Spring framework

Spring patří ve světě softwarového průmyslu k nejpoužívanějším frameworkům současnosti. Jedná se o open source (vydáván pod Apache Licence, Version 2.0[1]) framework použitelný v jakékoli javovské aplikaci (avšak hlavní uplatnění nachází ve webových podnikových aplikacích), je obklopen širokou uživatelskou a vývojářskou komunitou, je velmi intenzivně vyvíjen, navíc je napsán srozumitelně (ve smyslu self-explanatory) a má výjimečně kvalitně zpracovanou dokumentaci, přestože se bez ní v praxi vývojář ve většině případů obejde. Stačí mu totiž pouze zdrojový kód.

Celý framework se skládá z řady podprojektů soustředících se vždy na určitou problematiku, jednotlivé části jsou spolu integrovatelné a většina z nich je použitelná i samostatně. Příkladem je Spring Security. Spring Security, jak už z názvu vyplývá, řeší otázky zabezpečení aplikace. Projekt byl znám dříve jako Acegi security a jedná se o de facto standard. Byl dokonce portován i pro jiné platformy, například pro .NET.

Alternativním frameworkem podobného rozsahu je Seam společnosti RedHat (součást JBoss) a další alternativou mohou být referenční nástroje zaštítěné značkou Java Enterprise Edition. Referenční nástroje jsou bohužel stále těžkopádné a kvůli režii, která nutně předchází vydání každé další verze, jsou relativně zastaralé. Z toho důvodu byl jako aplikační framework zvolen Spring.

Nejvyužívanějšími částmi z rodiny Spring jsou Spring Core (poskytuje IoC container, AOP[2] nástroje), Spring Security a Spring Web Services.

[1] Apache License. *The Apache Software Foundation* [online]. 2004 [cit. 2013-04-20]. Dostupné z: http://www.apache.org/licenses/LICENSE-2.0.html .

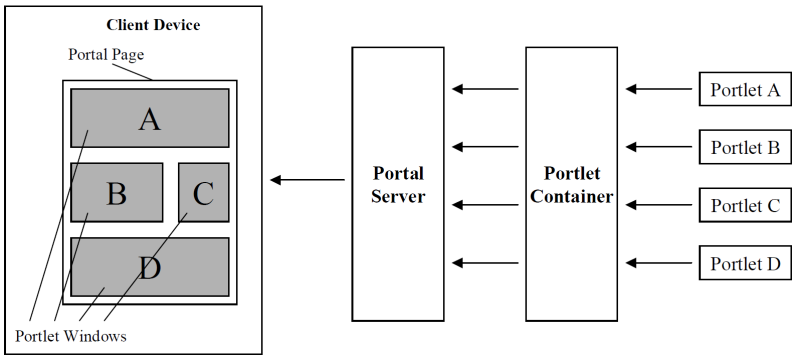
[2] AOP je zkratka pro Aspect Oriented Programming, což je programovací technika sloužící k separaci koncernů (anglicky 'separation of cross cutting concerns'). Typickým příkladem použití je logování. Kód související s logováním totiž téměř nikdy nesouvisí s kódem, kde se vyskytuje. Velmi zjednodušeně AOP umožňuje tento kód extrahovat a pomocí definovaných pravidel definovat, kdy se má volat, čímž velmi výrazně zvyšuje kvalitu kódu.

3.1.2 Hibernate ORM framework

Hibernate ORM je framework vyvíjený společností RedHat. Je implementací specifikace JPA[3]. Na rozdíl od aplikačního frameworku je zde použito standardní Java API a Hibernate byl zvolen z podobných důvodů jako Spring. Jde zejména o rozsáhlou komunitu a kvalitní dokumentaci včetně dostupných zdrojových kódů.

Framework poskytuje abstraktní vrstvu mezi databází (přesněji řečeno se nejedná o databázi, ale JDBC[4] API) a programem a tím umožňuje, aby software mohl být vyvíjen s maximálním využitím paradigmat a zvyklostí objektově orientovaného programování.

3.1.3 Liferay portal



Obrázek 3.1: Schéma portálové stránky podle JSR-286[5].

Jako uživatelské rozhraní slouží webový portál. Specifikace Java EE definuje, co je to Java Portál (dále portál) a Java Portlet v dokumentu JSR-286[5]. Portál je v zásadě webovou aplikací, která implementuje autentizační mechanizmy, nabízí možnost personalizace, správu a agregaci obsahu apod. Java Portál se v mnohém podobá populárnímu řešení CMS, mezi které patří například Wordpress nebo Joomla.

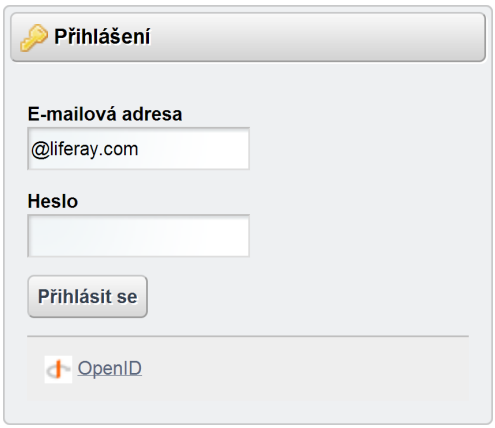
Hlavním stavebním prvkem portálu je Java Portlet (dále pouze portlet). Portlet je webová aplikace, která většinou slouží k jedinému účelu. Příkladem může být portlet pro registraci uživatele, přihlašovací portlet nebo portlet pro online podání zásilky v případě portálu např. České Pošty. Výstupem portletu je generovaný text, téměř vždy se jedná o (x)html. Jak může vypadat konkrétní portlet je vidět na obrázku 3.2. Na jedné stránce portálu může být několik portletů najednou. Portlety jsou určeny k běhu v Portlet Containeru (dále portlet kontejner).

[3]JSR 317: JavaTM Persistence 2.0. *Java Community Process* [online]. 2009 [cit. 2013-04-20]. Dostupné z: http://jcp.org/en/jsr/detail?id=317 .

[4]Java Database Connectivity - jedná se o API, které sjednocuje přístup k relačním databázím. Benefit využití tohoto API je v relativní nezávislosti kódu na použitém RDBMS. JDBC využívá k připojení adaptér, který dodává právě dodavatel databázového systému.

[5]JSR 286: Portlet Specification 2.0. *Java Community Process* [online]. 2008 [cit. 2013-04-20]. Dostupné z: http://jcp.org/en/jsr/detail?id=286

Portlet kontejner je běhové prostředí pro portlety, obstarává životní cyklus portletu a zprostředkovává komunikaci mezi dalšími portlety a portálem. Rozdíl mezi portlet kontejnerem a portálem je takový, že portál se stará čistě o zobrazení (dá se říci o agregování výstupů z jednotlivých portletů), zatímco kontejner slouží pouze jako běhové prostředí a vágně řečeno není vidět.



Obrázek 3.2: Ukázka jednoduchého portletu pro přihlášení.

Na trhu existuje řada produktů, které jsou implementací portlet kontejneru. V případě této aplikace jsou zajímavé hlavně ty produkty, které implementují jak portlet kontejner, tak jsou i samotným portálem. Hlavními dvěma kandidáty jsou JBoss Portal a Liferay. První zmíněný je využíván známým informačním systémem studijní agendy IS/STAG. Se systémem Liferay má autor práce praktickou zkušenost a verze Liferay Community Edition, která je zdarma i pro komerční použití včetně zdrojových kódů, plně vyhovuje nárokům na tento projekt, a to je důvodem, proč byl vybrán právě tento produkt.

3.1.4 JavaServer Faces

Jako prezentační technologie byl zvolen framework JavaServer Faces (dále JSF) ve verzi 2.1. Jedná se o nástupce JSP[6]. Framework je postaven nad Java Servlet a nabízí moderní způsob vytváření bohatých uživatelských rozhraní v prostředí webu. Mezi špičkovými programátory a softwarovými inženýry panují neshody ohledně architektonického zařazení frameworku, zda se jedná o MVP, nebo MVC. Podle názoru autora práce se jedná o MVP a přiklání se tak k názoru Jonase Bandiho[7], který tvrdí, že JSF odpovídají spíše vzoru MVP.

Výhodou JSF je návrh, který umožňuje jejich snadnou rozšiřitelnost. Využívá toho řada společností tak, že vyvíjejí komponentové balíky. Jedná se například o ADF vyvíjené společností Oracle, RighFaces, patřící do rodiny produktů JBoss (RedHat), a v neposlední řadě

[6]JSR 245: JavaServerTM Pages 2.1. *Java Community Process* [online]. 2013 [cit. 2013-04-20]. Dostupné z: http://jcp.org/en/jsr/detail?id=245

[7]Presentation Patterns: MVC vs. MVP (once again). *CLOSED-LOOP* [online]. 2010 [cit. 2013-04-20]. Dostupné z: http://blog.jonasbandi.net/2010/09/presentation-patterns-mvc-vs-mvp-once.html

PrimeFaces jako jeden z nejpopulárnějších JSF rozšíření současnosti. Komponentová sada PrimeFaces ve verzi 3.5 byla zvolena právě pro tento projekt. Komponenty jsou většinou stavěny nad existujícími knihovnami, ať už se jedná o klientské, jako je jQuery, bootstrap, YUI a nebo o serverové, například framework Atmosphere pro realizaci Comet, jenž je též nazýván jako reverse AJAX.

3.2 Architektura aplikace

Vzhledem k požadavku na škálovatelnost, možnost co nejsnazšího dalšího rozšiřování a úprav a povaze systému, která odpovídá SaaS je nutné, aby aplikace byla dělena na co nejvíce celků s velmi volnými vazbami (anglicky známo též pod pojmem loose-coupling). Tomuto požadavku nejlépe vyhovuje architektura SOA.

3.2.1 Systém jako celek

Systém je tedy rozdělen na dvě hlavní části, a to na webový portál (frontend) a aplikaci poskytující webové služby (backend). Výhodou tohoto řešení je velmi výrazné oddělení aplikační logiky od uživatelského rozhraní.

Frontend může být velice snadno nahrazen jinou webovou aplikací napsanou v jiném jazyce, s naprosto odlinym konceptem (třeba za pomoci Angular.js - MVC v browseru klientského počítače) a nebo pro úplně jinou platformu (například mobilní aplikace pro Android nebo desktopová aplikace pro MacOSX).

Backendová aplikace se stará o veškerou práci s daty. V aplikaci je implementováno vše, co se týká zabezpečení a řízení uživatelských práv. Dále aplikace agreguje byznys logiku a poskytuje rozhraní, jímž zpřístupňuje operace nad daty klientům.

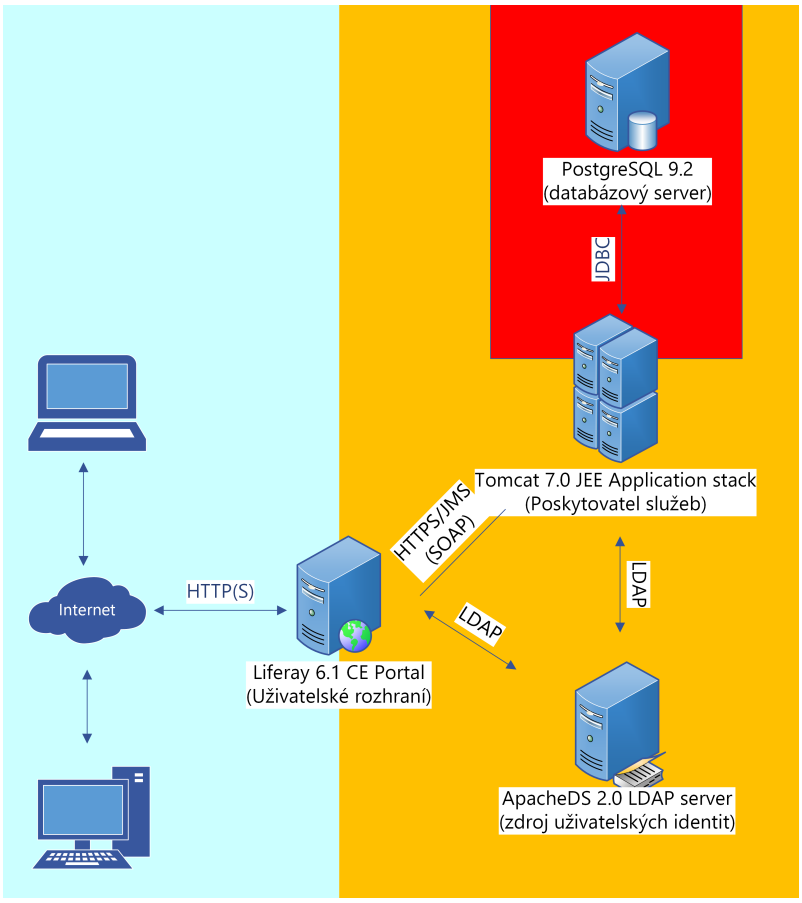
Komunikace mezi frontendem a backendem je realizována pomocí webových služeb protokoly SOAP a REST. Na obrázku číslo 3.3 je znázorněno, jak je systém navržen a rozčleněn.

3.2.2 Modularita

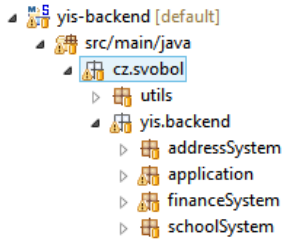
Systém se skládá z modulů, které se budou postupně upravovat, vyvíjet anebo odstraňovat. Moduly jsou na úrovni kódu reprezentovány balíky. Ukázka balíkové struktury je vidět na obrázku 3.4.

Jádro systému je umístěno v balíku *cz.svobol.yis.backend.application* a obsahuje API, které mohou ostatní moduly využívat. Kvůli co možná největšímu vynucení volných vazeb jsou moduly navrženy tak, aby každý balík, v případě že je to nutné, poskytoval maximálně jedno veřejné rozhraní. Tohoto bylo dosaženo díky *package-protected* modifikátoru přístupu. Toto je zásadní, proto tomu bude věnována větší pozornost.

V Javě existují čtyři typy modifikátorů přístupu. *Private* označuje vlastnost za privátní a umožňuje přístup jen z kontextu objektu, v němž je deklarována. *Protected* rozšiřuje kontext přístupnosti o potomky třídy objektu. *Package-protected* přístup (v Javě default)



Obrázek 3.3: Architektura systému.

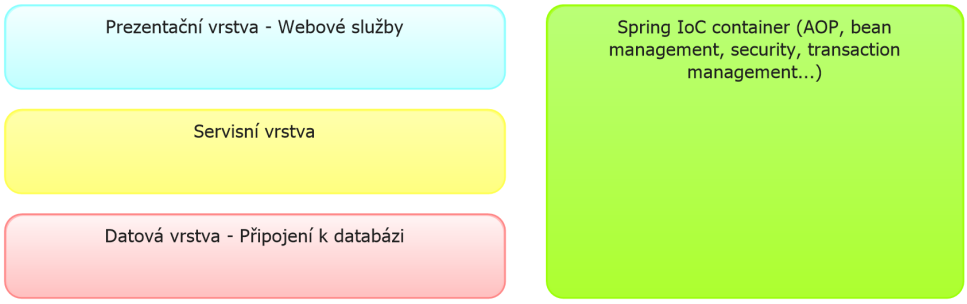


Obrázek 3.4: Balíková struktura - moduly.

znamená pole viditelné pouze objektům z téhož balíku. Tento modifikátor je velmi zajímavý, protože umožňuje například skrýt DAO (jedná se spíše o Table Data Gateway[1, p. 114]) objekty ostatním modulům a tudíž zvýšit zabezpečení dat. Jediný, kdo může DAO komponenty využívat je modul, v němž se nacházejí a pokud je třeba, aby ostatní moduly zacházely s daty tohoto modulu, musí pro to využít jediné veřejné API modulu. Autor práce se s tímto přístupem v praxi ještě nesetkal a použil jej poprvé. Posledním modifikátorem je *public*, který deklaruje proměnnou jako veřejnou. Jedná se o nejširší kontext. Modifikátory byly popisovány na proměnných, ale platí i pro metody.

Jádro systému nabízí jedno veřejné rozhraní *ApplicationService*, které mohou ostatní balíky využívat. Zbylé komponenty (kromě doménových objektů) jsou skryty. Tím je docíleno relativní udržitelnosti kódu, protože závislosti mezi moduly jsou jasně patrné a striktně definované.

3.2.3 Vrstvy aplikace



Obrázek 3.5: Vrstvy backandové části systému.

Backend aplikace, jak je vidět na obrázku 3.5, využívá běžnou třívrstvou architekturu skládající se z *prezentační*, *servisní* a *datové* vrstvy.

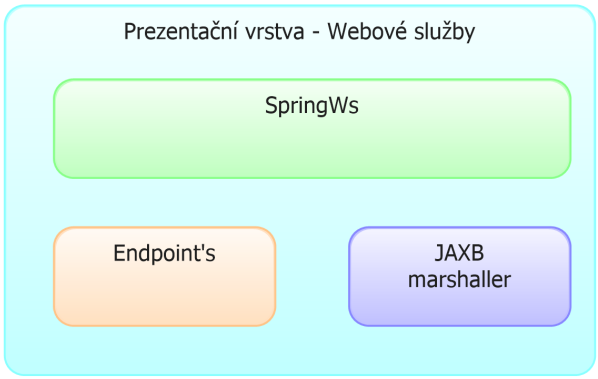
Na obrázku 3.6 jsou patrné hlavní části prezentační vrstvy backendu. Jedná se o framework Spring Web Services, který poskytuje základ pro tvorbu vlastních endpointů[8] a zjednodušuje odesílání a přijímání SOAP zpráv tím, že nabízí tzv. fasádu nad komponentami typu *MessageFactory*, *MessageSender* a také *Marshallery*[9].

Obrázek 3.7 znázorňuje komponenty datové vrstvy. Je zde prezentována abstrakce připojení k databázi, kde komponentou přímo komunikující s databází je JDBC. API JDBC využívá Hibernate pro připojení a komunikaci s databází. Hibernate abstrahuje také dotazování na databázi nástroji na generování SQL dotazů a o to víc odstiňuje programátora z objektového světa od relační databáze. Vrstva DAO objektů využívá Hibernate a

[8]Jako endpoint je u webových služeb nazývána komponenta, která se svou funkcí velmi podobá kontroleru v MVC.

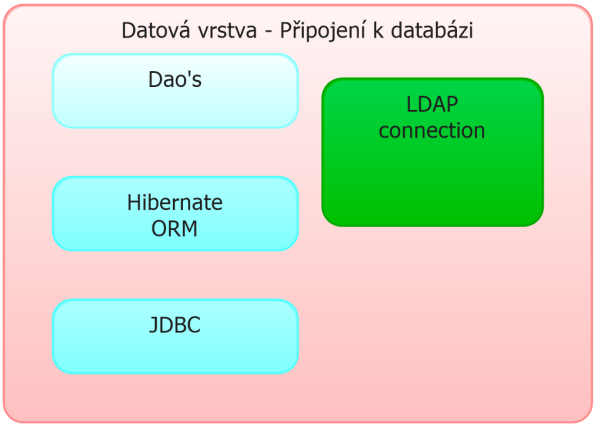
[9]Marshaller je softwarová komponenta starající se o převod datové reprezentace objektu v programu do tvaru vhodného pro transport nebo archivaci. V našem případě jde o převod doménových objektů aplikace na XML zprávy a naopak.

nabízí už konkrétní operace s daty. Příkladem může být metoda s takovouto signaturou *List<Person> findPersonsByLessonId(Integer lessonId);* která je využívána na servisní vrstvě.



Obrázek 3.6: Detail prezentační vrstvy backendu.

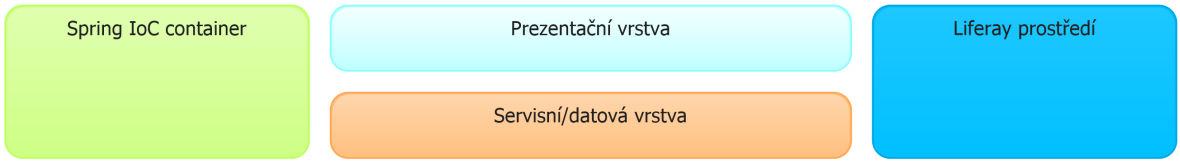
Servisní vrstva je ze všech tří vrstev nejtenčí. Jedná se o řadu servisních rozhraní a jejich implementací. V zásadě propojuje prezentační a datovou vrstvu. Nachází se zde řízení transakcí a volání metod je zabezpečené. To znamená, že volat servisní metodu může jen ten, kdo má odpovídající práva. Toho je docíleno použitím AOP, kdy před každým zavoláním metody probíhá kontrola práv přihlášeného uživatele a v případě nedostatečných oprávnění je vyhozena výjimka. Tím pádem se metoda nevykoná. Dalším bezpečnostním prvkem je již zmíněné řízení transakcí. Servisní metody označené jako *@Transactional* způsobí před započetím jejich vykonávání (opět docíleno pomocí technik AOP) vytvoření nové transakce. V případě, že je v průběhu vykonávání metody vyhozena vyjímka, transakce je automaticky zrušena a proveden *rollback*. V opačném případě se provede *commit*.



Obrázek 3.7: Detail datové vrstvy backendu.

Na obrázku 3.8 je znázorněno rozvrstvení frontendové části. Frontend by měl být relativně tenký, s minimem aplikační logiky. Neměla by se zde nacházet žádná byznys logika

a mělo by jít pouze o klienta backendu. Jediný úkol aplikace je poskytnout grafické rozhraní uživateli, v tomto případě (webová aplikace) vygenerovat kód, který bude možno klientským strojem interpretovat.



Obrázek 3.8: Vrstvy portálové aplikace.

3.3 Model systému

Tato sekce je zaměřena na podrobnější modelování systému. Jde zejména o databázové schéma, které je odvozeno z doménového modelu, a netriviální návrhové problémy, které musely být řešeny.

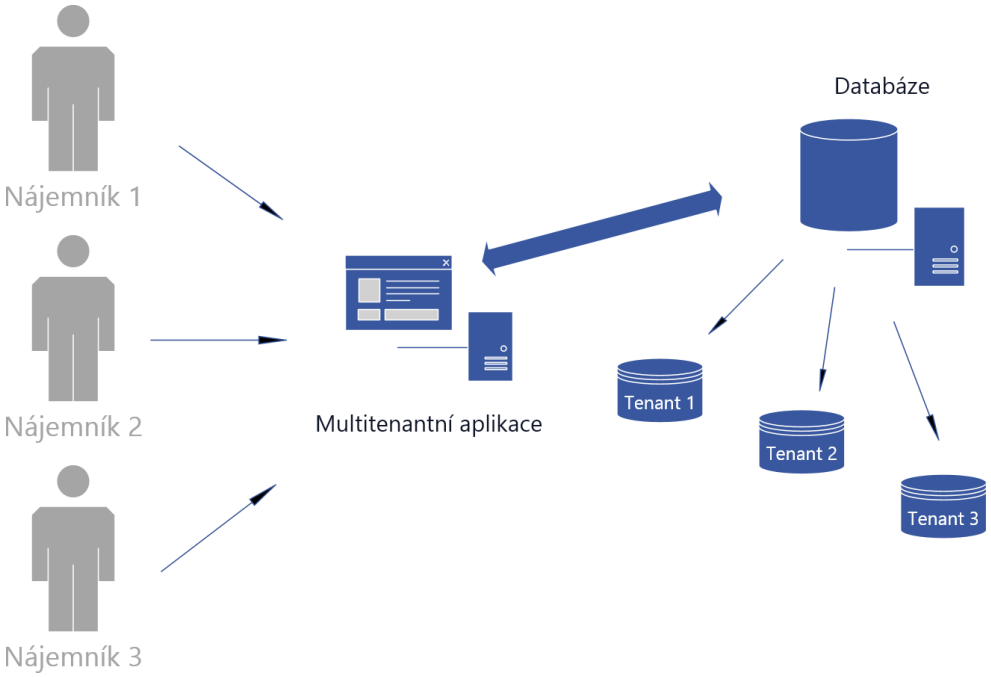
3.3.1 Multitenance

Multitenance (z anglického slova Multitenancy) je jedním z několika požadavků na architekturu systému. Velmi důležitá část multitenantí architektury je ukryta zejména na úrovni dat. Protože data jsou aktivem tenantů (nebo též nájemníků), tak v případě sdílené aplikace (což je hlavním předpokladem multitenance), je nutné data jednotlivých nájemníků oddělit. Modelový příklad multitenantní architektury je vidět na obrázku 3.9.

Oddělit data můžeme v zásadě dvěma hlavními způsoby. Prvním ze způsobů je, že každý tenant bude mít vlastní databázi nebo databázový stroj. Výhody jsou poměrně snadná škálovatelnost a velmi výrazné oddělení nájemníků (snadná obnova a záloha dat každého tenanta nezávisle). Mezi nevýhody patří zejména vyšší nároky na systémové zdroje a jistý technický problém nastává v případě, že je požadováno pracovat s daty všech tenantů.

Druhým způsobem je mít data všech tenantů v jedné databázi a každou tabulku obohatit o takzvaný diskriminátor. Diskriminátor je další sloupec v tabulce, který v sobě uchovává identifikátor nájemníka. Je možné pak rozeznat komu patří které záznamy. Mezi výhody patří menší náročnost na systémové zdroje, a to zejména tehdy, když systém používá mnoho tenantů, kde každý vlastní relativně málo dat. Další výhodou je (oproti prvnímu způsobu) možnost relativně snadno pracovat s daty více nájemníku najednou, protože pro každou entitu je použita vždy jen jedna tabulka. Oddělení dat je velmi slabé, záloha a obnovení dat nájemníků nezávisle na sobě je náročnější a riskantnější proces. Další nevýhodou jsou vyšší náklady na implementaci. V určitých případech se ale prvotní náklady na vývoj vyplatí.

Ostatní způsoby řešení multitenance se nacházejí mezi prvním a druhým výše zmíněným přístupem. Pro tento projekt byl zvolen přístup právě z této oblasti. Data jednotlivých



Obrázek 3.9: Příklad architektury multitenantní aplikace.

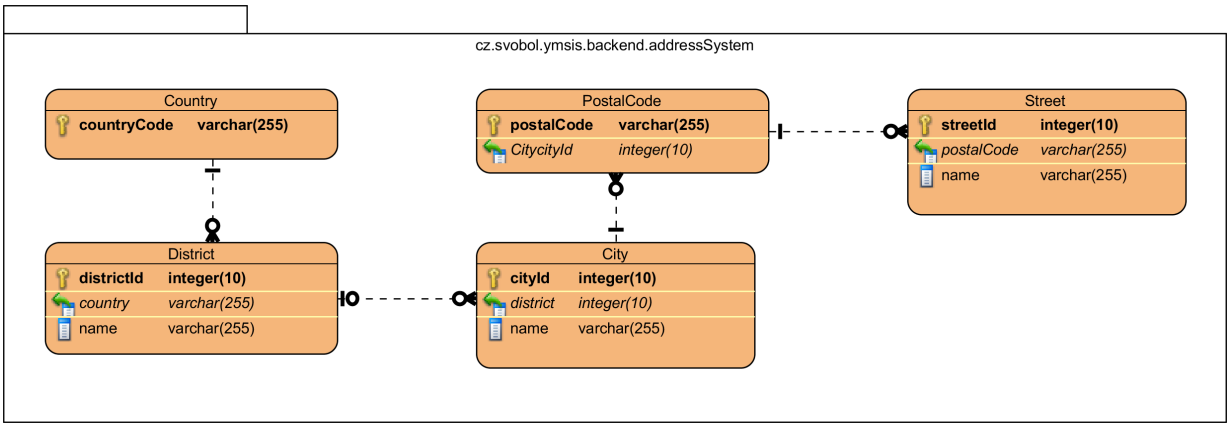
tenantů budou oddělena na základě různých databázových schémat v rámci jedné databáze. To znamená, že každému tenantovi bude vytvořeno vlastní databázové schéma s názvem shodným s jeho identifikátorem. Toto řešení kombinuje výhody a nevýhody obou hlavních přístupů a vzhledem k povaze tohoto projektu je ho možné považovat za vhodné.

3.3.2 Entitně relační diagram databáze

Entitně relační schéma je kvůli přehlednosti rozděleno na čtyři části. Jednotlivé části odpovídají konkrétním modulům systému.

Na diagramu 3.10 je vidět schéma adres. Systém adres je v samostatném modulu a je pro něj vyčleněno zvláštní schéma z toho důvodu, že v budoucnu se počítá s integrací s Registrem územní identifikace, adres a nemovitostí v České Republice a nebo s obdobným systémem v jiných zemích. Tento způsob řešení dovoluje pozdější validaci adres nebo usnadnění vyplňování zadáním pouze PSČ a ulice.

Diagram čislo 3.11 zobrazuje datovou základnu pro jádro aplikace. Fialovou barvou jsou vyznačeny nepřímo právnické subjekty, které jsou v systému ostatními moduly využívány. Toto na první pohled možná nesrozumitelné řešení vzniklo na základě požadavku, kdy je potřeba uzavírat právo platné smlouvy se zákazníky (žáky, nájemníky systému, kantory,atd.) a promítnout je do systému. Základem je entita *Contractor* která reprezentuje fyzickou osobu a figuruje tam, kde dochází k uzavíraní určité smlouvy (například zápis na hodinu). Entita *Company* využitím reference rozšiřuje *Contractor* a v systému modeluje právnickou osobu a fyzickou osobu podnikající. Na objektové úrovni (v kombinaci s ORM)



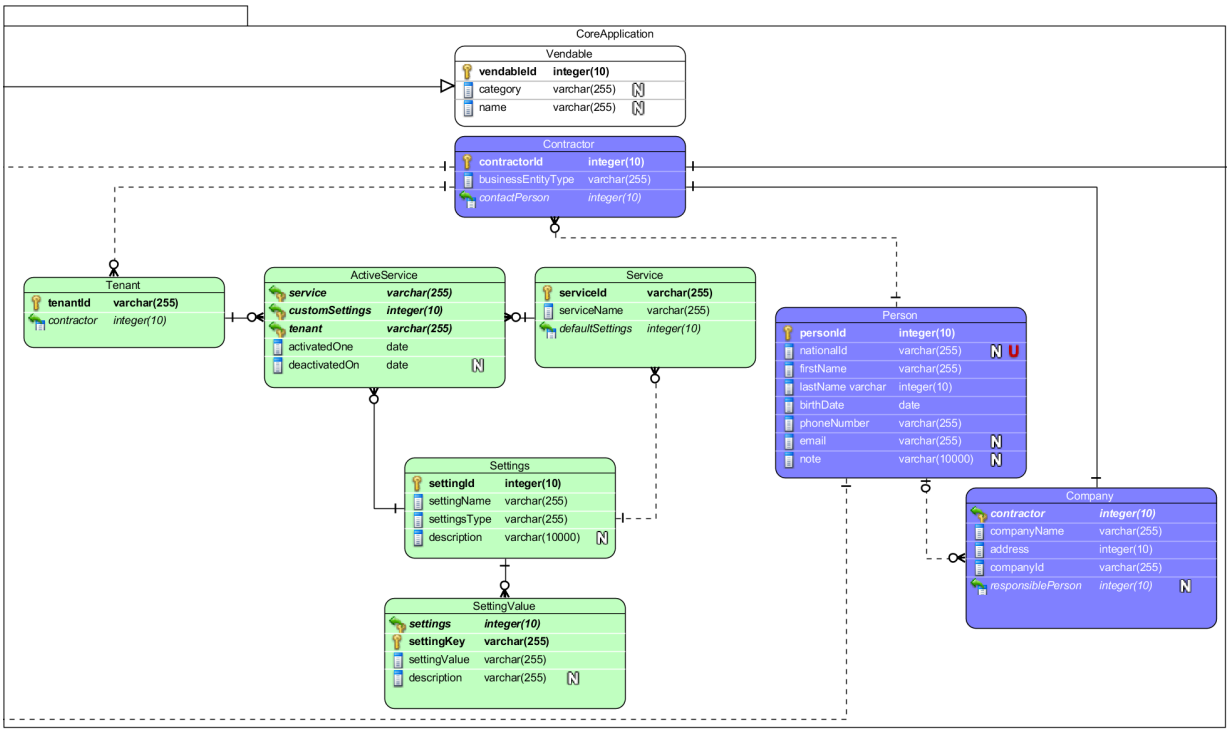
Obrázek 3.10: ER diagram systému adres.

je pak možné využít polymorfismu, což je velmi výhodné.

Entity vyznačené zelenou barvou jsou zvláštní tím, že spolu se systémem adres jsou jediné globální pro celou aplikaci. Ostatní entity jsou totiž kvůli multitenanci ve zvláštním databázovém schématu. Důvodem, proč jsou globální, je, že obsahují datovou základnu pro multitenanci samotnou. Jedná se o evidenci tenantů, služeb a modulů (a jejich nastavení) nabízených portálem. Dále je zde zaznamenáno, který tenant aktivoval jaké služby s případným uživatelským nastavením.

Na obrázku 3.12 je znázorněno schéma entit spojených se školami. Nejdůležitějšími entitami jsou *Lesson* modelující pravidelně vyučovanou hodinu a *Enrollment*, která představuje jeden zápis na hodinu. Reference *Enrollment* na *Vendable* je využita k realizaci dědičnosti, čili *Enrollment* je potomkem *Vendable* a tím pádem může být předmětem obchodu. *Branch* představuje pobočku, kterých tenant může provozovat několik. *Room* je představitelem učeben. *Course* nese informace o vyučovaných kurzech. Kurzem může být třeba hra na klávesy. *Program* je modelem učebního programu. V návaznosti na příklad kurzu, může programem být třeba Populární klavír 2 (název učebního programu hry populárních písní na klavír).

Relativně jednoduché schéma na obrázku 3.13 ukazuje jádro modulu evidence plateb (též nazýván jako finanční modul) na úrovni databáze. V případě, že má tenant aktivní finanční modul, pro všechny *Contractor* položky v kontextu tenanta existuje právě jeden *Account*. *Income* představuje jakýkoli příjem adresovaný na *Account*. *Income* vznikne například když rodič zaplatí školné za svého svěřence. *Outcome* vzniká tehdy, když je z *Account* proveden proveden příkaz k úhradě *Debt*, a nebo když se z *Account* vybírá přeplatek v hotovosti. *Debt* je reprezentace dluhu a pohledávky. *Debt* vzniká například když je žák zapsán do hodiny. Jednoduchost modulu na úrovni dat je vyvážena značnou složitostí na úrovni aplikační logiky.



Obrázek 3.11: ER diagram jádra aplikace.

3.3.3 Komunikace mezi moduly

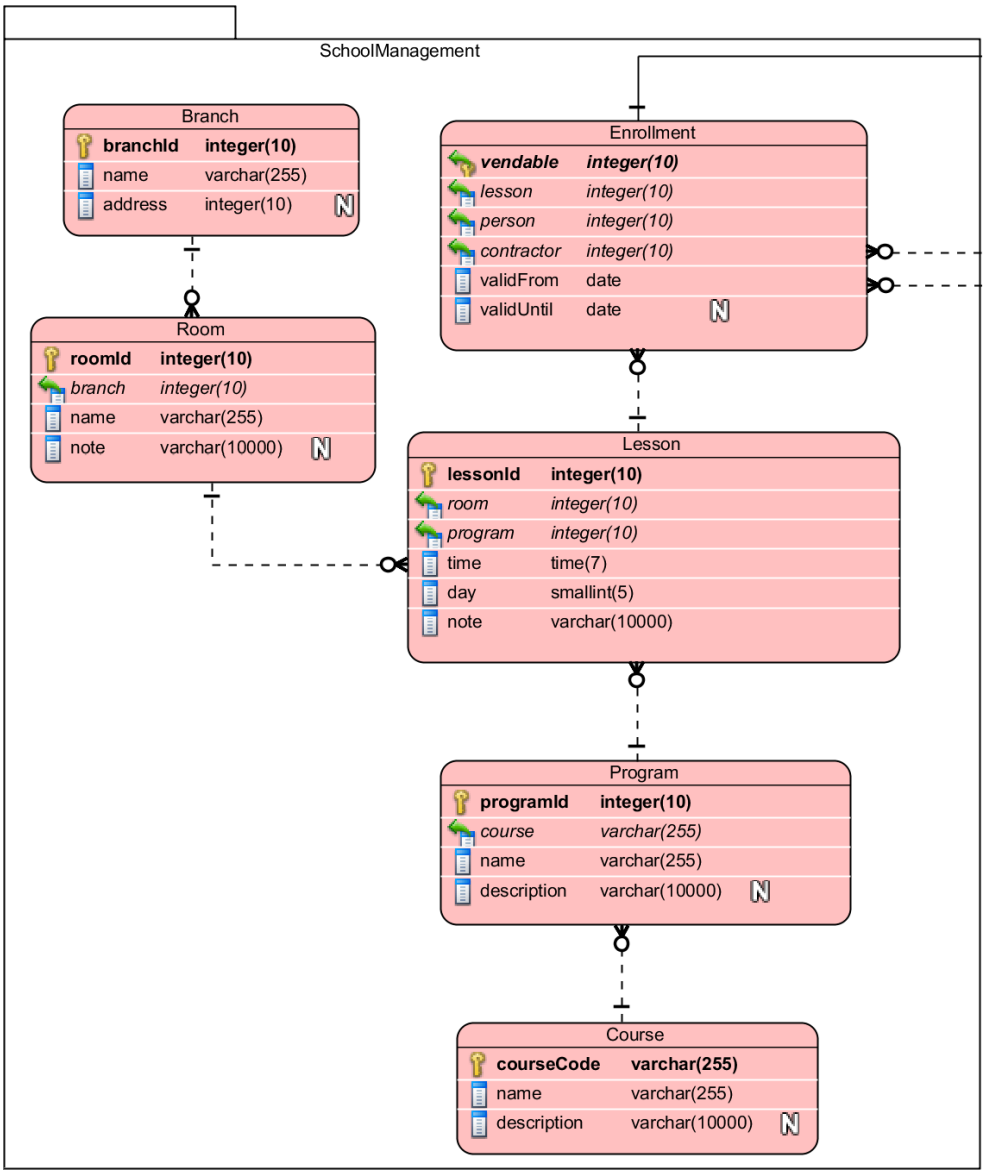
Jak již bylo několikrát zmíněno, při návrhu byl kladen velký důraz na co nejvolnější vazby mezi komponentami. Předpokládejme, že modul A je jádrem systému, B a C jsou rozšiřující moduly. Pak platí tyto výroky:

- Modul B nezávisí na modulu C a naopak.
- Modul A nezávisí na žádném jiném modulu.
- Modul B i C mohou záviset (a pravděpodobně i závisí) na modulu A.

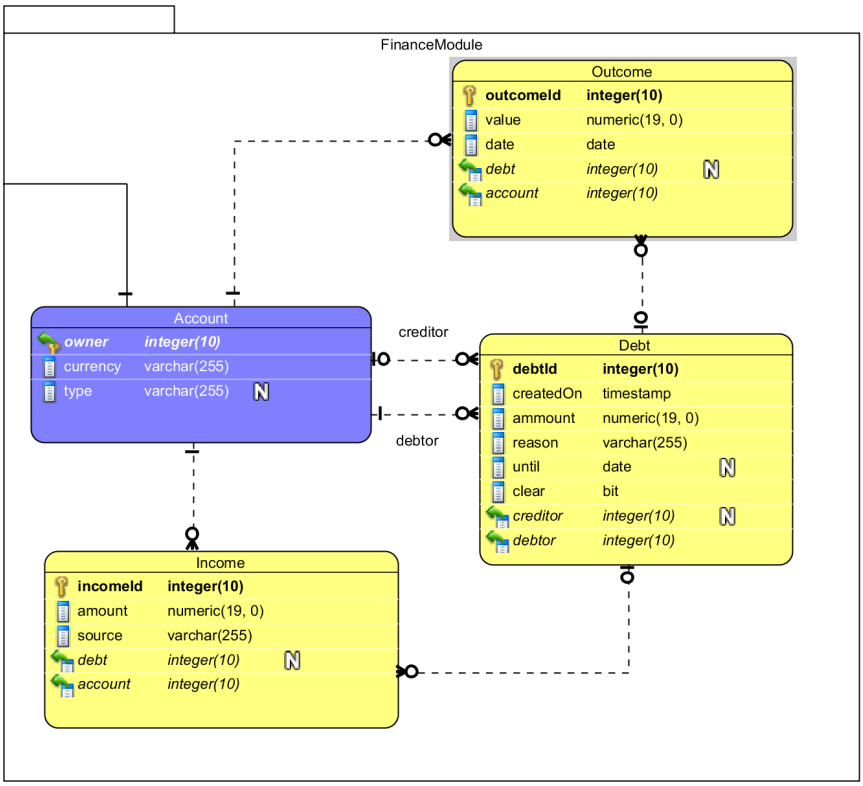
Výše uvedené ale představuje problém v případě, že je potřeba aby po zapsání žáka do hodiny v modulu *School* vznikl *Debt* v modulu *Finance*, a když jádrem aplikace je *Application*. Porušení pravidel nepřipadá v úvahu a tento druh komunikace modulů mezi sebou byl vyřešen použitím návrhového vzoru, který je velmi dobře znám pod anglickým názvem *Observer pattern*.

Na obrázku 3.14 je vyobrazeno konkrétní použití modifikovaného *Observer pattern* pro komunikaci mezi moduly. Diagramy jednotlivých tříd a rozhraní jsou pro tento účel zjednodušeny. Na diagramu jsou pouze prvky nutné k demonstraci komunikace po provedení nového zápisu.

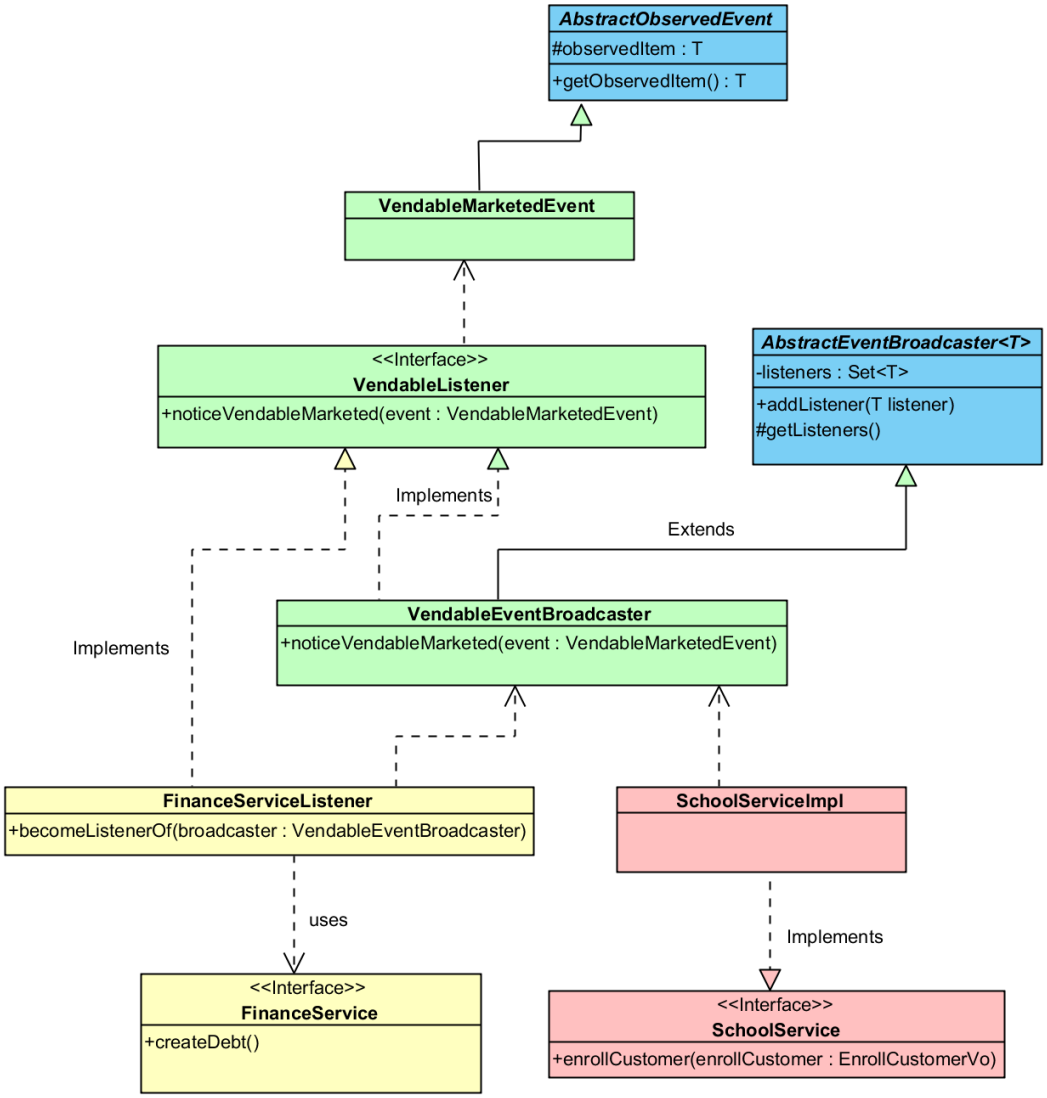
Modře vyznačené abstraktní třídy obsahují základ pro libovolnou implementaci komunikace. Jendá se o nadtřídu broadcaster, jejíž funkcí je agregovat posluchače události a



Obrázek 3.12: ER diagram modulu spojeného se školou.



Obrázek 3.13: ER diagram modulu evidence plateb a dluhů.



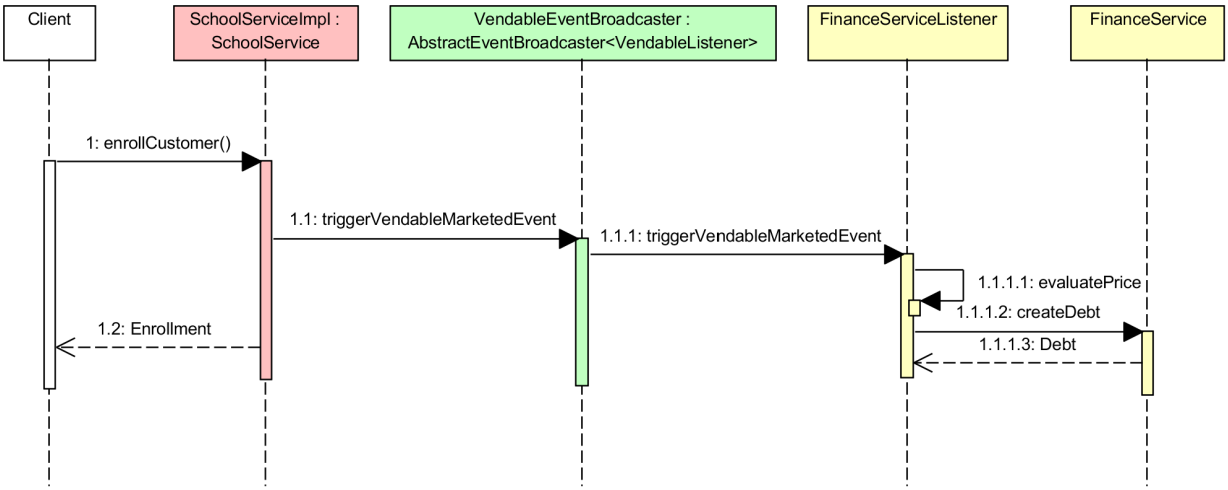
Obrázek 3.14: Diagram tříd realizace komunikace mezi moduly.

zároveň událostem naslouchat. Po zaznamenání události ji deleguje na ostatní registrované posluchače.

Prvky vyznačené zeleně jsou specifické pro údalosti spojené s objektem *Vendable*. V našem případě se jedná o zápis (*Enrollment*). Hlavním prvkem v této skupině je rozhraní *VendableListener*. Rozhraní deklaruje události které mohou nastat v souvislosti s objekty typu *Vendable*. Rozhraní je implementováno jak broadcasterem, tak komponentou *FinanceServiceListener*.

Žlutě jsou vyznačeny specifické typy modulu *Finance*. *FinanceServiceListener* naslouchá událostem delegovaným broadcasterem a eventuelně na ně reaguje. Například pokud *VendableEventBroadcaster* vyvolá událost *VendableMarketed*, *FinanceServiceListener* se postará o to aby byl případně vytvořen *Debt*.

Červenou barvou jsou vyobrazeny opět typy z rozšiřujícího modulu, tentokrát jde o modul *School*. V tomto případě je modul zdrojem události. Jakmile dojde k vykonání metody *SchoolServiceImpl#enrollCustomer()* , vytvořením zápisu je vyvolána událost na objektu broadcaster, který ji případně deleguje. Schéma komunikace je vidět na obrázku 3.15.



Obrázek 3.15: Sekvenční diagram komunikace modulů při zápisu žáka.

Poslední zajímavostí je způsob inicializace tohoto celku. Je k tomu využito *Dependency Injection*, kdy rozsah platnosti (známo jako *Scope*) objektů *FinanceServiceListener*, *SchoolServiceImpl* a *VendableEventBroadcaster* je v rámci celé aplikace (tzv. *ApplicationScope*). Do *FinanceServiceListener* je při inicializaci dodán IoC kontejnerem *VendableEventBroadcaster* a posluchač pak sám sebe registruje do dodaného broadcasteru. Obdobně je tomu u *SchoolServiceImpl* kde ale nedochází k registraci, ale služba si na broadcaster drží referenci aby mohla vyvolávat události.

Z předchozího je patrné jak velmi silná separace mezi moduly je a že nedošlo k její narušení ani v tomto případě. Vyplývá z toho několik zásadních důsledků. Když odstraníme červenou část (například smažeme celý modul), systém bude nadále funkční a ostatní části nebudou ovlivněny. Totéž platí o žluté části. Implementace takto striktní nezávislosti se ukazuje v případě systému tohoto charakteru jako zásadní výhoda.

Kapitola 4

Implementace prototypu

Tato kapitola se věnuje implementaci prototypu systému, nebo také můžeme říct platformy, nad kterou bude systém stavěn.

4.1 Vývojové prostředí

Jako vývojové prostředí je použit nástroj Spring Tool Suite ve verzi 3.2.0 vyvíjenou společností springsource. Jedná se o modifikaci známého prostředí Eclipse verze 3.8.x. Prostředí usnadňuje vývoj aplikací využívajících Spring framework a také opravuje některé chyby nacházející se ve standardní Eclipse.

4.2 Zásady vývoje

Pro co nejsnazší srozumitelnost je kód psán konvenčně. Takový kód, který dodržuje zdánlivě nepodstatné zásady o počtech mezer, odřádkováních, formátování komentářů či zápisu výrazů apod., může být považován za kvalitnější a čitelnější. Psaní jednotvárného konvenčního kódu je požadavkem u tohoto projektu.

Další důležitou zásadou je tvorba self-explanatory kódu. V projektu není nutné veškerý kód dokumentovat (to ovšem neplatí pokud se jedná o veřejná rozhraní, neboli API), duležité ale je psát kód srozumitelně. Znamená to pojmenovávat metody tak, aby už z názvu bylo patrné jaký je jejich účel, pojmenovávat proměné tak, aby bylo zřejmé co je jejich obsahem. Naprosto nepřípustné jsou nic neříkající názvy jako *attr*, *arg*, *var*, apod.

Na obrázku 4.1 je krátká ukázka kódu z aplikace. Z ukázky je patrná většina konvencí. Jedná se o implementaci rozhraní *FinanceService*. Za povšimnutí stojí deklarace třídy *FinanceServiceImpl* na řádce s číslem 23. Třída není veřejná, ale je *package-protected*, to znamená, že mimo balík (modul) neni viditelná. Nicméně rozhraní které implementuje, je věřejné. V praxi je zde vidět návrh, který byl popsán v kapitole 3.2.2 Modularita.

Aplikace je programována proti rozhraním, znamená to, že pokud je v systému komponenta *PriceListDao*, pak bude existovat rozhraní se stejným názvem a většinou bude existovat alespoň jedna třída, která rozhraní implementuje. Předpokládejme, že budeme

mít dvě implementace. Jedna bude ceníky načítat z .csv souborů a druhá z databáze. Pak pojmenování implementujících tříd podle konvencí bude například takovéto: *CsvPriceListDaoImpl* a *DbPriceListDaoImpl*. Nyní je vidět podstat rozhraní. Ostatní komponenty pracující s ceníkem by neměly být závislé na konkrétní implementaci, ale jen na abstrakci (rozhraní v tomto případě deklaruje kontrakt).

Dalším důvodem je AOP. Pokud je IoC kontejner požádán o *PriceListDao*, předpokládá se, že vrátí jednu z implementací (záleží na konfiguraci). Žádná komponenta nezávisí na konkrétní implementaci, proto nezáleží, která implementace je kontejnerem vrácena. To znamená, že kontejner může vrátit místo jedné z našich implementací proxy objekt, který zprostředkuje volání implementace. Uvnitř proxy objektu pak má kontejner možnost vykonávat vlastní kód před a po vykonání vlastní metody. Tento mechanizmus je v projektu používán především k řízení transakcí a zabezpečení volání metod.

Programátor pracuje na určité části systému. Preferován je způsob vývoje známý pod anglickým názvem *test-driven development* (dále TDD). Využívá zásady popsané výše, kdy bylo definováno programování proti rozhraním. Rozhraním se v zásadě definuje chování, následně se napíše unit test, který testuje požadovanou funkcionalitu. Vágně řečeno to způsobí, že testy budou selhávat do doby, než bude existovat korektní implementace. Tento způsob umožňuje rozdělit vývoj na stále se opakující malé celky, a to celý proces usnadňuje a zefektivňuje. Testování se věnuje podrobněji kapitola 6 Testování.

```java
/**
 *
 * @author Lukáš Svoboda
 */
@Service
class FinanceServiceImpl implements FinanceService {

    @Autowired
    private AccountDao accountDao;
    @Autowired
    private IncomeDao incomeDao;
    @Autowired
    private OutcomeDao outcomeDao;
    @Autowired
    private DebtDao debtDao;


    /*
     * (non-Javadoc)
     * @see cz.svobol.yis.backend.financeSystem.FinanceService#findAccounts(cz.svobol.yis.backend.financeSystem.Account)
     */
    @Transactional(readOnly=true)
    public List<Account> findAccounts(Account account) {
        notNull(account, "Account argument cannot be null.");
        List<Account> result = new ArrayList<Account>();
        if (account.getAccountId() != null) {
            result.add(accountDao.findById(account.getAccountId()));
        } else {
            result.addAll(accountDao.findByExample(account));
        }
        return result;
    }
```

Obrázek 4.1: Ukázka kódu třídy *FinanceServiceImpl*.

4.3 Nástroje

4.3.1 Verzování a řízení projektu

Kvůli ne úplně malému rozsahu projektu, je nutné spravovat verze. Jako verzovací systém byl zvolen Mercurial. Jako poskytovatel webového repositáře slouží služba bitbucket[1]. Pro řízení projektu je použit webový open source systém Redmine[2].

4.3.2 Apache Maven

Maven je nástroj pro správu projektu (myšleno na úrovni zdrojových kódů). Mezi jeho nedocenitelné vlastnosti patří schopnost spravovat závislosti na knihovnách třetích stran (nebo též závislosti). Pokud má kód nějaké závislosti, musí tyto být na classpath v době spouštění programu. V případě javovských aplikací je běžná závislost na několika desítkách knihoven. Manuální správa je v takovém případě velmi náročná a neefektivní. Maven umí knihovny stahovat a automaticky zařazovat do projektu z internetových repozitářů. Dokáže též vyřešit konflikty mezi verzemi závislostí.

Další úlohou Mavenu při vývoji je kompilace a sestavování (buildování) projektů například do .war (Web application Archive[3]) archivů.

Maven v neposlední řadě umožňuje použití nepřeberného množství pluginů, sloužících k nejrůznějším úkolům. Na tomto projektu se nejvíce používají pluginy pro generování zdrojových kódu. Jedná se o generování tříd z XSD schémat, o generování QueryDSL pomocných tříd, a generování SQL databázových schémat.

Důležité je, že všechny tyto procesy je možné díky mavenu automatizovat a provést je pomocí jediného příkazu. Veškerá konfigurace nastavení Mavenu se nachází v souboru s názvem *pom.xml*. Mavenovský projekt se typycky skládá ze složky *src* obsahující zdrojové kódy a souboru *pom.xml* s nastavením mavenu.

4.3.3 QueryDSL

QueryDSL je framework pro generování typově bezpečných SQL dotazů. Je možné ho použit s Hibernate (generování HQL dotazů). Framework si vygeneruje pomocné třídy z JPA oanotovaných doménových objektů a ty se pak používají pro konstrukci dotazů místo toho aby se SQL dotaz konstruoval přímo v podobě řetězce. Není proto potřeba každou část SQL příkazu, která přišla ze vstupu pracně kontrolovat a zároveň nehrozí riziko SQL injection. Použití frameworku QueryDSL je ukázáno na obrázku 4.2. Použití frameworku QueryDSL by mělo být jednoduché a pochopitelné pro všechny, kteří umí používat standardní SQL.

[1] *Bitbucket* [online]. 2013 [cit. 2013-04-20]. Dostupné z: https://www.bitbucket.org/

[2] Redmine je webová aplikace sloužící pro řízení projektů a bug tracking. Využívá technologii Ruby on Rails. Webové stránky projektu: http://www.redmine.org

[3] JSR 154: JavaTM Servlet 2.4 Specification. *Java Community Process* [online]. 2007 [cit. 2013-04-20]. Dostupné z: http://jcp.org/en/jsr/detail?id=154

```
19      @Override
20      public Collection<Enrollment> findByPersonId(Integer personId) {
21          HQLQuery query = new HibernateQuery(getCurrentSession());
22          QEnrollment enrollment = QEnrollment.enrollment;
23          return query
24                  .from(enrollment)
25                  .where(enrollment.attendee.person.personId.eq(personId))
26                  .list(enrollment);
27      }
28
```

Obrázek 4.2: Ukázka použití frameworku QueryDSL.

4.3.4 Běhová prostředí

Pro provoz backendu byl využit Servlet Container Apache Tomcat 7.0.35. Frontend je nasazen taktéž na serveru Apache Tomcat, ale ve verzi 7.0.27. Standalone databázový server byl použit PostgreSQL 9.2 a pro testování a vývoj je využíván HyperSQL in-memory databázový stroj. Druhý zmíněný je napsán kompletně v Javě, to znamená, že je velmi snadno použit v případě testování. Adresářový server byl zvolen ApacheDS ve verzi 2.0.0M10. Jedná se taktéž o kompletně javovskou aplikaci a tudíž je možné jej také využít při běhu testů.

4.4 Implementace

Systém se na úrovni kódu skládá ze tří projektů. Všechny projekty jsou spravované nástrojem maven. Projekt s názvem *yis-backend* obsahuje zdrojový kód backendové aplikace. V projektu s názvem *yis-frontend* se nacházejí zdrojové kódy k portletům jež jsou použity k sestavení uživatelského rozhraní v portálu. Třetí projekt obsahuje xml soubory popisující webové služby (čili wsdl soubory) a schémata datových typů používaných v při přenosu protokolem SOAP. Název projektu je *yis-schema*.

První dva zmíněné projekty závisejí na projektu *yis-schema*, protože používají pro komunikaci SOAP. V projektu je definován maven plugin pro vygenerování JAXB2 tříd z xml a ty jsou pak v programech využívány. Vygenerované třídy obsahují anotance nutné pro serializaci marshallery.

Kapitola 5

Agilní vývoj

Jak bylo poznamenáno v úvodu, po implementování funkční platformy systému bude vývoj pokračovat dle agilního manifestu[1]. Konečná podoba systému není známa a je pravděpodobné, že systém se bude velmi výrazně měnit. Tomu odpovídá architektura systému popsaná v kapitole číslo 3.2 Architektura aplikace. Agilní metodika vývoje umožňuje provádět poměrně zásadní změny v návrhu a požadavcích i v pozdních fázích vývoje, a to je podnětem pro její použití.

Podstatou agilního vývoje na tomto projektu je co nejrychlejší vývoj a profilace produktu. Vývoj probíhá následovně:

- *KDYKOLI* Uživatel systému nebo realizátor navrhne změnový požadavek (dále issue) a zadá ho do systému Redmine. Vzniká tzv. product backlog.
- *KDYKOLI* Vývojářský tým složený z uživatelů systému a realizátorů ohodnotí (dále realizační tým) změnové požadavky byznys hodnotou a implementační náročností.
- *PŘED ZAČÁTKEM SPRINTU* Realizační tým vybere ze požadavků v systému ty, které budou v tomto springu implementovány. K výběru může sloužit poměr mezi hodnotou a náročností. Dochází k vytvoření tzv. sprint backlogu.
- *V PRŮBĚHU SPRINTU* Vývojář prezentuje implementované funkcionality a změny v systému uživateli. Uživatel (klient) implementaci případně připomínkuje.
- *PŘED KONCEM SPRINTU* Provede se zhodnocení sprintu, co bylo implementováno a co nikoli. Po akceptačním testování se vydává nová verze software.

Na obrázku 5.1 je ukázka product backlogu. Jedná se o snímek části obrazovky ze systému Redmine. Na obrázku 5.2 je vidět detail issue. V tomto případě se jedná i požadavek na vytvoření finančního modulu. Z obrázku je patrné, že realizace jedné issue může připomínat metodiku vodopád.

[1] *Manifesto for Agile Software Development* [online]. 2001 [cit. 2013-04-24]. Dostupné z: http://agilemanifesto.org/

#	Tracker	Status	Priority	Subject	Assignee	Updated
172	Úkol	New	Normal	Vytvorit zapisovaci dialog.	Lukas Svoboda	04/07/2013 01:06 pm
166	Požadavek	New	Low	K telefonnímu číslu přidat před input lokalizovanou předvolbu (+420)		03/16/2013 11:49 pm
164	Úkol	New	Normal	Contact a vse kolem nej do Base	Lukas Svoboda	03/13/2013 06:24 pm
163	Úkol	New	Normal	Jedna ze tříd má v názvu Addess... místo Address	Lukas Svoboda	03/13/2013 06:25 pm
162	Bug	New	Normal	Uložení hodiny nepřepíná do bound a hodinu nejde vyhledat	Lukas Svoboda	03/13/2013 04:13 pm
161	Bug	New	Normal	po uložení pobočky je potřeba obnovit stránku, jinak pobočka není vidět při vytváření hodin	Lukas Svoboda	03/13/2013 04:09 pm
160	Bug	New	Normal	Po uložení zaměstnance (člověka) zmizí mail a datum narození	Lukas Svoboda	03/13/2013 04:08 pm
159	Bug	New	Low	Při přidávání adres v bound se nepřepíná do modified	Lukas Svoboda	03/13/2013 03:45 pm
156	Úkol	In Progress	Normal	Implementace financniho modulu		03/13/2013 06:14 pm
153	Požadavek	In Progress	Normal	Financni modul		03/16/2013 11:51 pm
152	Úkol	New	Normal	Service Provider modul		03/06/2013 09:26 pm
151	Úkol	New	Normal	CRM modul		03/07/2013 03:56 pm
150	Úkol	New	Normal	Analyticky modul		03/07/2013 03:58 pm
149	Úkol	New	Normal	GUI pro spravu uctu		03/06/2013 09:24 pm
148	Úkol	New	Normal	System dochazky		03/07/2013 03:59 pm
147	Úkol	New	Normal	Modifikace pro multitenance		03/06/2013 09:22 pm
145	Bug	Resolved	Normal	Nově vytvoření zaměstnanci se neobjevují při pidávání hodiny	Lukas Svoboda	03/09/2013 02:31 am
144	Bug	New	Normal	Po zakliknutí programu se nezobrazuje jeho kurz	Lukas Svoboda	03/06/2013 04:07 pm
143	Bug	Resolved	Normal	Nově vytvořené kurzy se nezobrazují v nabídce kurzu při přidávání programu	Lukas Svoboda	03/09/2013 02:31 am
142	Bug	Resolved	Normal	Nefunguje mazání kurzů	Lukas Svoboda	03/09/2013 02:31 am
141	Bug	Resolved	Normal	Další bugy v BOUND u zaměstnanců	Lukas Svoboda	03/09/2013 02:30 am
140	Bug	New	Normal	V BOUND stavu u zaměstnance chybí tlačítko smazat	Lukas Svoboda	03/06/2013 03:57 pm

Obrázek 5.1: Snímek product backlogu ze systému Redmine.

Požadavek #153

Update | Log time | Watch | Copy | Delete

Financni modul

« Previous | 10 of 31 | Next »

Added by Admin Yamaha about 1 month ago. Updated less than a minute ago.

Status:	In Progress	**Start date:**	03/07/2013
Priority:	Normal	**Due date:**	
Assignee:	-	**% Done:**	73%
Category:	-	**Spent time:**	33.60 hours
Target version:	-		
Business value:	10	**Story points:**	15

Subtasks Add

~~Úkol #154~~: Analyza financniho modulu	Closed	
~~Úkol #155~~: Design financniho modulu	Closed	
Úkol #156: Implementace financniho modulu	In Progress	

Related issues Add

Obrázek 5.2: Detail issue v systému Redmine.

Kapitola 6

Testování

Testování je nedílnou součástí vývoje každého software. Testování může být prováděno v několika úrovních. Po úrovní je možno si představit jak velkou část softwaru testujeme. V případě tohoto projektu jsou nejdůležitjší jednotkové (též nazývané jako unit) testy, integrační testy a testování aplikace uživatelem.

6.1 Unit testy

Unit testy se testují nejmenší části aplikace. Jedná se o třídy nebo metody. Tento typ testů se dá velmi dobře automatizovat. Unit testy píše programátor v průběhu vývoje a v případě TDD, na jeho samém počátku. V projektu je využit framework JUnit 4.

```
17 public class EnrollmentDaoTest extends BaseDaoTest {
18
19     @Autowired
20     private EnrollmentDao enrollmentDao;
21     @Autowired
22     private AttendeeDao attendeeDao;
23
24
25     @Test
26     public void testInsert() {
27         Enrollment enrollment = new Enrollment();
28         enrollment.setName("test");
29         Attendee attendee = attendeeDao.findById(new Attendee.AttendeeId(3, 2));
30         enrollment.setAttendee(attendee);
31         enrollment.setValidFrom(new Date());
32         enrollment.setValidUntil(DateTime.now().plusDays(90).toDate());
33         Enrollment merged = enrollmentDao.merge(enrollment);
34         assertTrue(merged.getType() == VendableType.ENROLLMENT);
35         assertTrue(merged.getId() != null);
36     }
37
38 }
```

Obrázek 6.1: Ukázka unit testu třídy *EnrollmentDao*.

Aby mohl být kód testovaný, musí být především testovatelný. Špatně testovatelný kód je takový kód, který například závisí na *ServletRequest* nebo jiném objektu který se těžko

nahrazuje v testovacím prostředí. Obecně se dá tvrdit, že pokud je dodržena atomicita (nedělitelnost) metod (čili metodá slouží k jednomu jedinému jasnému účelu), tak je velká pravděpodobnost vytvoření kvalitního kódu.

Při testování kód vyvíjený proti rozhraním (jak bylo popsáno v sekci 4.2 Zásady vývoje) benefituje ze snadné nahraditelnosti komponent. Například pokud bude potřeba otestovat DAO objekt, není vhodné používat produkční nebo vývojářskou databázi. Pravděpodobně by byla porušena zásada testování a sice opakovatelnost a nebo nezávislost na běhovém prostředí. Proto je výhodnější použít in-memory databázi, která se inicializuje a spustí vždy současně s testem. V případě závislosti na abstraktních rozhraních, může být komponentě pak místo produkční databáze dodána in-memory neboli takzvaný mock. Techniky mockování (nebo také nahrazování) se při testování a vývoji velmi často využívá.

Na obrázku 6.1 je ukázka jednoduchého unit testu. Test se pokusí vytvořit nový zápis, uložit ho do databáze a pak ověří, že zápis byl opravdu uložen.

6.2 Integrační testy

Integrační testy testují komplexnější část systému. Většinou se používají k testování, zda spolu více komponent komunikuje a zda se celek chová podle očekávání. Integrační testy byly v případě tohoto projektu použity zejména k testování webových služeb a klientů webových služeb.

```
7  public class AddressSystemEndpointTest extends BaseWsTest {
8
9      @Test
10     public void testGetCountries() {
11         StringSource sentMessage = new StringSource(
12             "<getCountries xmlns='http://www.svobol.cz/yis/schema'>" +
13             "</getCountries>"
14         );
15         StringSource expectedMessage = new StringSource(
16             "<getCountriesResponse xmlns='http://www.svobol.cz/yis/schema'>" +
17                 "<countries>" +
18                     "<countryCode>a</countryCode>" +
19                 "</countries>" +
20                 "<countries>" +
21                     "<countryCode>cz</countryCode>" +
22                 "</countries>" +
23                 "<countries>" +
24                     "<countryCode>de</countryCode>" +
25                 "</countries>" +
26                 "<countries>" +
27                     "<countryCode>p</countryCode>" +
28                 "</countries>" +
29                 "<countries>" +
30                     "<countryCode>sk</countryCode>" +
31                 "</countries>" +
32             "</getCountriesResponse>"
33         );
34         sendAndExpect(sentMessage, expectedMessage);
35     }
```

Obrázek 6.2: Ukázka integračního testu třídy *AddressSystemEndpoint*.

Ukázka kódu integračního testu na obrázku 6.2 evokuje dojem vyšší složitosti. V kódu jsou deklarovány a inicializovány těla dvou SOAP zpráv. Tělo odesílané zprávy a tělo očekávané odpovědi. Metoda *sendAndExpect()* volaná na řádku 34 způsobí odeslání zprávy na mock endpoint, provedení veškeré aplikační logiky jako by tomu bylo v případě běhu na serveru a po obdržení odpovědi srovná očekávanou zprávu s tou ve skutečnosti přijatou.

Test jako takový je v tomto případě simulovaným klientem webové služby a ten pak může komponentu (Endpoint) otestovat jako celek. Obdobným způsobem byly testováni klienti v projektu *yis-frontend*.

6.3 Testování systému uživatelem

Poslední druh testování, kterému bude věnována pozornost je testování systému jako takového uživatelem. Zde se už nejedná u automatizované testování strojem, ale člověkem. Tento druh testování lze také automatizovat, jde už ale o netriviální záležitost.

Vzhledem k vývojovému stádiu projektu a k tomu, jak dynamickými změnami systém momentálně prochází, nebylo tomuto věnováno příliš mnoho pozornosti. Po ustálení ale budou ale definovány tzv. testovací scénáře. Scénáře se budou týkat hlavních funkcionalit systému, jako například uložení nového žáka do systému, zapsání žáka do hodiny apod. Scénáře volně navazují na případy užití.

Testovací scénář bude definovat jak se má systém chovat nebo v jakém má být stavu když uživatel provede určitou akci. Testování pak bude probíhat v tom stylu, že tester (člověk, který provádí testování) bude procházet jednolivé scénáře a zaznamenávat výsledky do tzv. test reportu. Report bude sloužit jako podklad pro bugfix a analytické účely. Toto bude probíhat vždy před koncem iterace, což se předpokládá že bude jednou za měsíc.

Kapitola 7

Závěr

Tato práce vycházela ze zjištěné potřeby inovace nástrojů podporujících hlavní procesy v síti hudebních škol Yamaha, které se vyznačovaly především značnou nekoncepčností a roztříštěností.

V rámci práce pak byl navržen a částečně implementován IS na úrovni pokrývající základní funkcionality vyžadované hudebními školami. IS se v době odevzdání práce nachází v předprodukční fázi. IS je momentálně v ověřovacím provozu na jedné ze spolupracujících škol.

Záměrem řešitele je nejdéle počátkem příštího školího roku (9/2013) nabídnout systém do ostrého provozu na dalších školách. O jeho dalším vývoji bude rozhodnuto na základě odezvy uživatelů a na základě finanční rozvahy. Výhledově lze uvažovat o vytvoření klientských aplikací pro mobilní zařízení, zobecnění stávajícího modelu nasazení a rozšíření systému tak, aby byl použitelný i mimo síť Yamaha.

Seznam obrázků

Literatura

[1] *FOWLER, Martin.* **Patterns of enterprise application architecture.**
Boston: Addison-Wesley, c2003, xxiv, 533 p. ISBN 03-211-2742-0.

[2] *MULARIEN, Peter.* **Spring Security 3: secure your web applications against malicious intruders with this easy to follow practical guide.**
Birmingham, U.K.: Packt Open Source, 2010, x, 397 p. ISBN 978-1-847199-74-4.

[3] *GEARY, David M a Cay S HORSTMANN.* **Core JavaServer faces: secure your web applications against malicious intruders with this easy to follow practical guide.**
3rd ed. Upper Saddle River, NJ: Prentice Hall, c2010, xx, 636 p. ISBN 01-370-1289-6.

[4] *DAIGNEAU, Robert a Cay S HORSTMANN.* **Service design patterns: fundamental design solutions for SOAP/WSDL and RESTful Web services.**
3rd ed. Upper Saddle River, NJ: Addison-Wesley, c2012, xxv, 321 p. Addison-Wesley signature series. ISBN 03-215-4420-X.

[5] *SARIN, Ashish a Cay S HORSTMANN.* **Portlets in action: fundamental design solutions for SOAP/WSDL and RESTful Web services.**
3rd ed. Shelter Island, NY: Manning, c2012, xxvii, 612 p. Addison-Wesley signature series. ISBN 19-351-8254-4.

[6] *BINILDAS, C a Cay S HORSTMANN.* **Service oriented architecture with Java: using SOA and web services to build powerful Java applications.**
1st ed. Birmingham: Packt Publishing, 2008, xv, 258 p. Addison-Wesley signature series. ISBN 978-184-7193-216.

[7] *HERNANDEZ, Michael J.* **Návrh databází.**
1. vyd. Praha: Grada, 2006, 408 s. ISBN 80-247-0900-7.

[8] *PHILIP KOTLER, Gary Armstrong.* **Principles of marketing.**
12th ed. Upper Saddle River, N.J: Pearson/Prentice Hall, 2008, xxix, 661, [77] s. ISBN 01-361-3237-5.

Webová sídla

[1] *Specifikace na serveru Java Community Process*
`http://www.jcp.org`

[2] *Materiály na serveru Microsoft*
`http://msdn.microsoft.com`

[3] *Dokumentace na serveru Oracle*
`http://www.oracle.com`

[4] *Dokumentace na serveru Spring source community*
`http://www.springsource.org`

[5] *Dokumentace na serveru Hibernate*
`http://www.hibernate.org`

[6] *Dokumentace na serveru Liferay*
`http://www.liferay.com`

[7] *Dokumentace na serveru PostgreSQL*
`http://www.postgresql.org`

[8] *Dokumentace na serveru PrimeFaces*
`http://www.primefaces.org`

Příloha A

Seznam použitých zkratek

API Application Programming Interface

AOP Aspect-oeriented programmign

AJAX Asynchronous Javascript And XML

CMS Content Management System

CSV Comma-Separated Values

DAO Data Access Object

DB Data Base

ER Entity-relationship

HQL Hibernate Query Language

HTML HyperText Markup Language

IoC Inversion of Control

IS Information System

IT Information Technology

JAXB Java Architecture for XML Binding

JDBC Java DataBase Connectivity

JMS Java Messaging Service

JPA Java Persistence API

JSF JavaServer Faces

JSP JavaServer Pages

JSR Java Specification Request

LDAP Lightweight Directory Access Protocol

MVC Model View Controller

MVP Model View Presenter

ORM Object-relational Mapping

PHP Hypertext Preprocessor

RDBMS Relational DataBase Management System

REST Representational State Transfer

SaaS Software as a Service

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol

SQL Structured Query Language

TDD Test-driven Development

UC Use Case

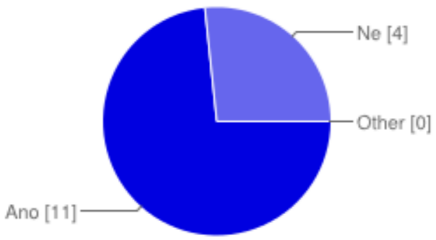
WSDL Web Services Description Language

XML Extensible Markup Language

XSD XML Schema Definition

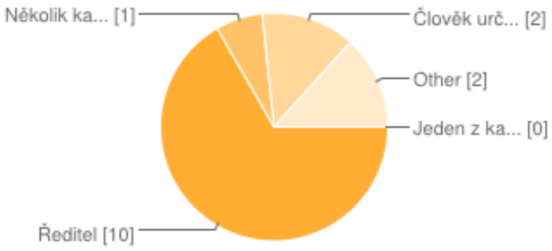
Příloha B

Data z dotazníků



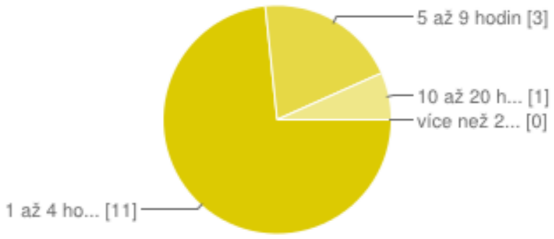
Ano	**11**	73%
Ne	**4**	27%
Other	**0**	0%

Obrázek B.1: Otázka: Má Vaše škola vlastní webové stránky?



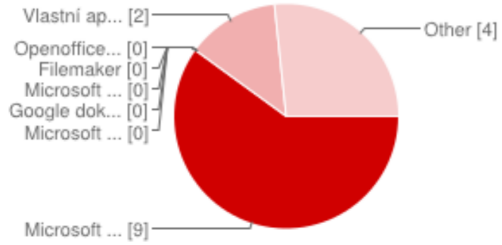
Jeden z kantorů	**0**	0%
Ředitel	**10**	67%
Několik kantorů	**1**	7%
Člověk určený speciálně pro tento účel	**2**	13%
Other	**2**	13%

Obrázek B.2: Otázka: Kdo se stará o agendu?



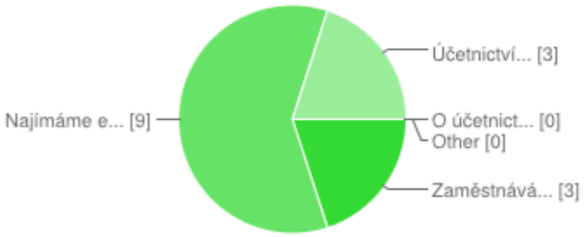
1 až 4 hodiny	**11**	73%
5 až 9 hodin	**3**	20%
10 až 20 hodin	**1**	7%
více než 20 hodin	**0**	0%

Obrázek B.3: Otázka: Kolik času týdně přibližně zabere agenda?



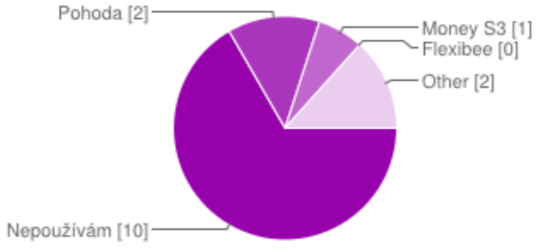
Microsoft Office Word a Excel	**9**	60%
Openoffice / Libreoffice	**0**	0%
Filemaker	**0**	0%
Microsoft Access	**0**	0%
Google dokumenty	**0**	0%
Microsoft office 365	**0**	0%
Vlastní aplikaci	**2**	13%
Other	**4**	27%

Obrázek B.4: Otázka: Pro agendu spojenou s evidencí žáků a plateb používáte?



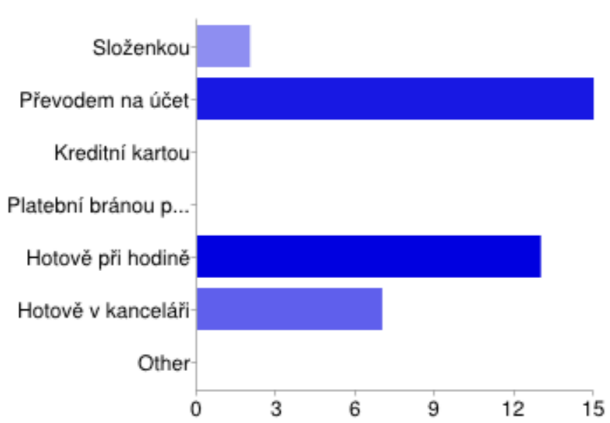
O účetnictví se stará jeden z kantorů	**0**
Zaměstnáváme účetního	**3**
Najímáme externí firmu	**9**
Účetnictví má na starost ředitel	**3**
Other	**0**

Obrázek B.5: Otázka: Jakým způsobem vedete účetnictví?



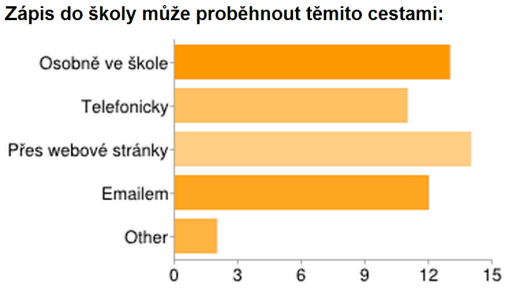
Nepoužívám	**10**	67%
Pohoda	**2**	13%
Money S3	**1**	7%
Flexibee	**0**	0%
Other	**2**	13%

Obrázek B.6: Otázka: Používáte účetní software? Jaký?



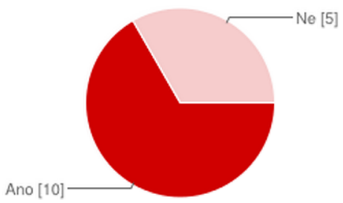
Složenkou	**2**	5%
Převodem na účet	**15**	41%
Kreditní kartou	**0**	0%
Platební bránou přes internet	**0**	0%
Hotově při hodině	**13**	35%
Hotově v kanceláři	**7**	19%
Other	**0**	0%

Obrázek B.7: Otázka: Jakými způsoby je možné uhradit školné?



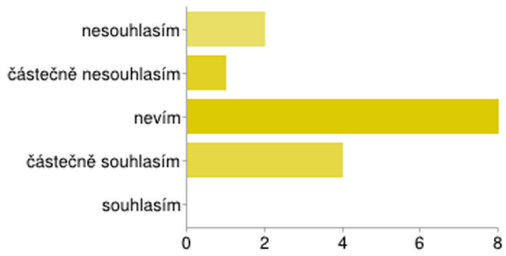
Osobně ve škole	**13**	25%
Telefonicky	**11**	21%
Přes webové stránky	**14**	27%
Emailem	**12**	23%
Other	**2**	4%

Obrázek B.8: Otázka: Jak způsoby je možné se zapsat do školy?



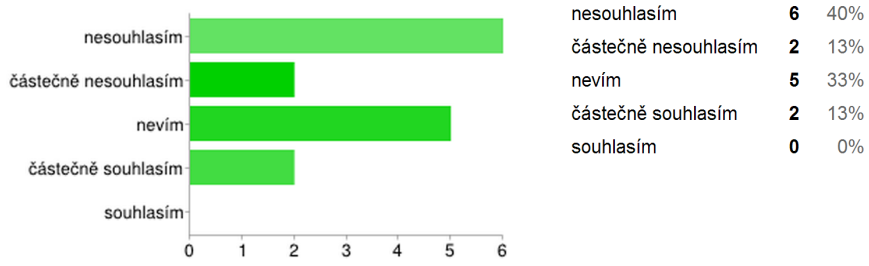
Ano	**10**	67%
Ne	**5**	33%

Obrázek B.9: Otázka: Zvažovali jste v minulosti použití informačního systému?

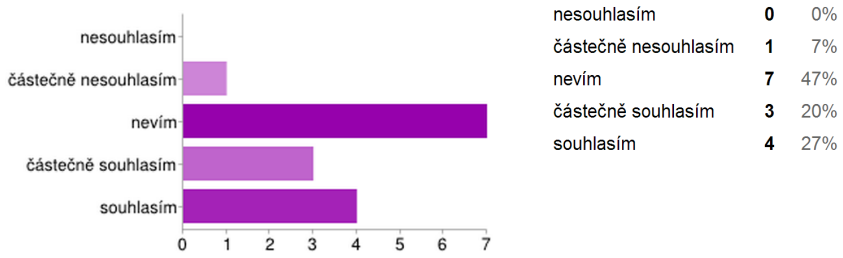


nesouhlasím	**2**	13%
částečně nesouhlasím	**1**	7%
nevím	**8**	53%
částečně souhlasím	**4**	27%
souhlasím	**0**	0%

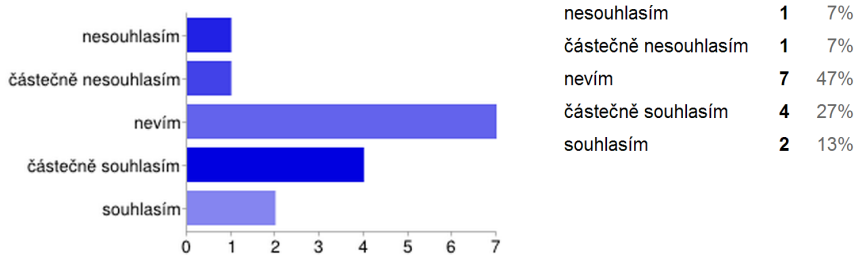
Obrázek B.10: Jak se ztotožňujete s výrokem: Informační systémy jsou drahé.



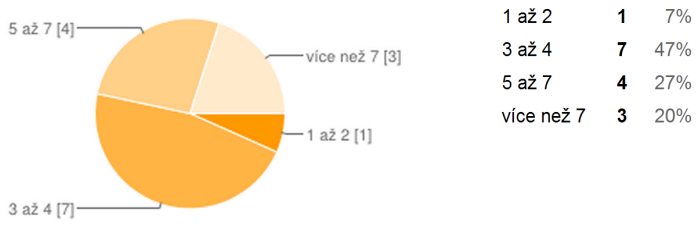
Obrázek B.11: Jak se ztotožňujete s výrokem: Informační systémy jsou složité.



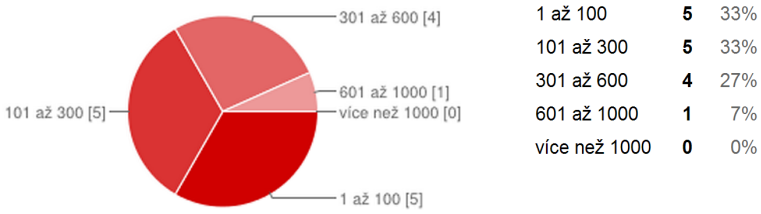
Obrázek B.12: Jak se ztotožňujete s výrokem: Informační systémy jsou spolehlivé.



Obrázek B.13: Jak se ztotožňujete s výrokem: Žádný IS nenabízí přesně to, co potřebuji.



Obrázek B.14: Otázka: Kolik učebních programů se ve Vaší škole vyučuje?



Obrázek B.15: Otázka: Kolik žáků řádově navštěvuje Vaši školu?

Příloha C

Scénáře případů užití

Případ užití	**YIS-001 - Přidat osobu**
Popis	Uživatel potřebuje přidat do systému osobu.
Aktéři	Uživatel Systém
Vstupní podmínky	Uživatel je přihlášen do systému a má potřebná oprávnění k přídávíní osob. Uživatel byl přesměrován na obrazovku.
Výstupní podmínky	

Hlavní scénář:

1. Systém zobrazí formulář pro zadání údajů o osobě.
2. Uživatel formulář vyplní a kliknutím na tlačítko uložit údaje odešle.
3. Systém podle odeslaných dat uloží do databáze novou osobu.
4. UC končí.

Zvláštní případy:

3.a Osobu nebylo možné uložit:

1. Systém zobrazí chybovou zprávu a umožní uživateli změnit zadané hodnoty a proces zopakovat.

Výskyt	Velmi častý

Případ užití	**YIS-002 - Přidat kontraktora**
Popis	Uživatel potřebuje zavést zákazníka do systému jako kontraktora, aby mohl s ním mohl uzavírat obchody a jiné smlouvy.
Aktéři	Uživatel Systém
Vstupní podmínky	Uživatel je přihlášen do systému a má dostatečná práva k přidávání kontraktorů. Uživatel byl automaticky přesměrován, či akci manuálně vyvolal, a nachází se na obrazovce přidat kontraktora.
Výstupní podmínky	

Hlavní scénář:

1. Systém uživatele vyzve k zadání informací o kontraktorovi a přiřazení kontaktní osoby.
2. Uživatel do systému zadá potřebné informace a vyhledá kontaktní osobu.
3. Uživatel potvrdí zadané údaje a systém provede uložení nového kontraktora do databáze.
4. UC končí

Zvláštní případy:

2.a Kontaktní osoba není v systému:

1. *KDYŽ* má uživatel dostatečná práva pro přidání osoby, *INCLUDE* **YIS-001 - Přidat osobu**.
2. *JINAK* systém odkáže uživatele na administrátora.

3.a Kontraktora nebylo možné uložit:

1. Systém zobrazí uživateli chybovou zprávu a umožní mu upravit zadané údaje a proces zopakovat.

Výskyt	Častý

Případ užití	**YIS-003 - Zapsat žáka**
Popis	Uživatel provede zapsání žáka do hodiny
Aktéři	Uživatel Systém
Vstupní podmínky	Uživatel je přihlášen do systému a má práva provádět operace zápisu.
Výstupní podmínky	

Hlavní scénář:

1. Uživatel otevře stránku zápisů.
2. Systém zobrazí úvodní stranu s nabídkou hodin a osob.
3. Uživatel vybere v levé straně obrazovky osobu a na pravé straně hodinu a vyselektuje ji.
4. V momentě, kdy jsou obě položky vybrané systém zobrazí tlačítko zapsat.
5. Uživatel klikne na tlačítko a systém přepne na obrazovku uzavření smlouvy a kde vyzve uživatele k výběru kontraktora a případně k zadání dalších informací.
6. Uživatel vyhledá kontraktora a doplní informace.
7. UC končí

Zvláštní případy:

6.a Kontraktor v systému neexistuje:

1. *KDYŽ* má uživatel dostatečná práva, *INCLUDE* **YIS-002 - Přidat kontraktora**.
2. *JINAK* systém odkáže uživatele na administrátora.

Výskyt	Častý

Případ užití	**YIS-004 - Změna zápisu osoby**
Popis	Uživatel potřebuje změnit zápis.
Aktéři	Systém Uživatel
Vstupní podmínky	Žák je zapsán do hodiny, uživatel je přihlášen do systému a má právo měnit zápisy.
Výstupní podmínky	

Hlavní scénář:

1. Uživatel vyhledá a vybere osobu
2. Systém zobrazí všechny hodiny na které je žák zapsán
3. Uživatel vybere hodinu a klikne upravit zápis
4. Systém zobrazí dialogové okno s dvěma volbami *Změnit zápis* a
5. *Zrušit zápis.*
6. Uživatel zvolí požadovanou akci.
7. UC končí

Zvláštní případy:

6.a Uživatel vybere *Zrušit zápis.*

    1. Systém vyzve

6.b Uživatel vybere *Změnit zápis.*

Výskyt	Méně častý

Případ užití	**YIS-005 - Vyúčtování služeb informačního systému nájemci**
Popis	Koncem zúčtovacího období je třeba vykalkulovat fakturované položky.
Aktéři	Systém
Vstupní podmínky	Nájemce využívá služeb informačního systému.
Výstupní podmínky	Vytvořený závazek nese dále zpracovatelné fakturační položky.

Hlavní scénář:

1. Systém vytvoří fakturační položky tak, že vyúčtuje používání jednotlivých modulů.
2. Na základě fakturačních položek systém zachytí pohledávku a vytvoří závazek nájemci.
3. UC končí

Zvláštní případy:
N/A

Výskyt	častý